Simple Metaheuristics for Scheduling

An empirical investigation into the application of iterated local search to deterministic scheduling problems with tardiness penalties

Am Fachbereich Informatik
der Technischen Universität Darmstadt
vorgelegte und genehmigte

Dissertationsschrift

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

von

Matthijs Leendert den Besten

aus Woerden
in den Niederlanden

Referent: Prof. Dr. Wolfgang Bibel
Korreferent: Dr. Thomas Stützle

Tag der Einreichung: 25. August 2004
Tag der mündlichen Prüfung: 6. Oktober 2004

Darmstadt 2004
Hochschulkennziffer D17

Zusammenfassung

Diese Dissertation behandelt die Konfiguration stochastisch lokaler Suchverfahren für kombinatorische Optimierungsprobleme mittels neuer, automatischer experimentellen Prozeduren. Die Konfiguration and Selektion des Suchverfahrens wird aufgrund einer genau spezifizierten, vom Rechner durchgeführten Evaluation der Performanz von verschiedenen Varianten des Suchverfahrens mittels Racingverfahren bestimmt. In Abhängigkeit des Aufwands der Erzeugung von Lösungen, werden die Lösungsgüte und der Berechnungsaufwand, d.h. die Rechenzeit zur Erzeugung der Lösungen, unterschiedlich berücksichtigt. Können Lösungen mit ungefähr gleichem Aufwand generiert werden, ist die Güte der Lösungen das Hauptkriterium für die Auswahl. Gibt es erhebliche Unterschiede im Aufwand der Erzeugung von Lösungen, wird die Rechenzeit explizit in der Auswahl berücksichtigt. Falls die Suchverfahren eine Serie von Lösungen steigender Güte generieren können, wird die Güte jeder neuen solchen Lösung zusammen mit der entsprechenden Rechenzeit im Auswahlverfahren berücksichtigt. Die Anwendung dieser Auswahlverfahren wird in dieser Dissertation anhand von Konstruktionsheuristiken, Varianten von iterierten Verbesserungsverfahren und der iterierten lokalen Suche für deren jeweilige Anwendung auf zwölf unterschiedliche Schedulingprobleme dargestellt. Das Endergebnis ist jeweils ein Suchverfahren, dessen Performanz oft dem “State–of–the–Art” im Problembereich entspricht. Zusätzlich konnten durch eine eingehende, statistische Analyse der Versuchsdaten neue Einsichten in den Einfluss der Komponenten von Suchverfahren auf die Performanz in Abhängigkeit der Struktur der untersuchten Optimierungsprobleme gewonnen werden.

Preface

> Faire de la bonne cuisine demande un certain temps. Si on vous fait attendre, c'est pour mieux vous servir, et vous plaire.
>
> Restaurant Antoine, New Orleans.[1]

This dissertation is many things: It is an investigation into the effect of problem specification on algorithm design. It is also a description of the procedures to be followed in order to effectively configure and calibrate approximate algorithms for combinatioral optimization problems. Last but not least, it is a report on the attainment of state–of–the–art performance with relatively simple algorithms on relatively complex problems.

Most of the research for this dissertation was done in the office at Alexanderstraße. Right opposite this office, there is a pizzeria called Da Nino. Whenever the Intellectics group had something to celebrate, Da Nino would be the place to go. Not only did Da Nino serve excellent food at those occasions, the restaurant also provided inspiration for the topic of this dissertation.

Da Nino served as a source of inspiration in two respects. First of all, Da Nino is a natural habitat for the scheduling problems that I was attacking in my research. When the chef of the kitchen devises a plan to handle the pizza orders in such a way that the customers are served swiftly, he is solving a scheduling problem. Secondly, the composition of menus and the perfection of recipes is not unlike the development of algorithms. In both cases, components are added or removed from the whole and the whole is evaluated by a third party — customers in the case of a restaurant, an objective function in the case of an algorithm.

While this dissertation draws from Da Nino for its inspiration, the research that it reports on extends well beyond this particular area of application. Scheduling problems occur everywhere in industry and commerce and the need for systematic testing and design of algorithms occurs everytime one tries to come up with a procedure to solve these scheduling problems and other combinatorial optimization problems.

The contribution of this dissertation is threefold:

1. Algorithms are developed for a wide variety of closely related scheduling problems. Deeper insight is obtained with respect to the effect of problem

[1] Source: [Broo 95]

specification on algorithm behavior through the comparison of algorithm development and performance for these specific problems.

2. Algorithm development involves a series of systematic experiments to test a wide variety of algorithm instantiations. Deeper insight is obtained with respect to the effect of sequencing, iteration, and neighborhood specification on algorithm performance through this extensive testing and the analysis of these tests.

3. The algorithms' assessment and analysis is done according to a well–defined procedure. Deeper insight is obtained with respect to algorithm development through the definition and application of this procedure for assessment and analysis.

The dissertation is composed of three parts and an appendix. Part I contains the background information. It has two chapters. Chapter 1 provides an introduction to scheduling problems and ways to solve these problems and Chapter 2 offers an exposition of the procedures that are followed in the development and assessment of the algorithms. Part II constitutes the core of the dissertation. In Part II, the development of approximate algorithms to the scheduling problems of choice is described in detail. This part has three chapters. Chapter 3 describes the design and development of methods to construct scheduling solutions from scratch; Chapter 4 dissects local search methods in order to find out which components compound to the best algorithm; and Chapter 5 delineates in what way the repetition of local search after the initial construction of a solution can be done advantageously. Finally, Part III comprises benchmark results for the state–of–the–art algorithms developed in the dissertation in Chapter 6 and concludes with an overview of the lessons learned from the algorithm development and assessment in Chapter 7. The appendix contains tables with detailed results of the research described in Parts II and III.

Acknowledgments

I would like to thank the people at the Intellectics group at the TU Darmstadt, and also the people at IRIDIA and the people at Fonseca's group for their support and comments. In particular, I would like to thank Thomas Stützle, Wolfgang Bibel, Mauro Birattari, Luis Paquete, Carlos Fonseca and Susana Fernandez. In addition, I would like to thank the people at Oxford and Surrey who provided relief from the thesis work and various stages of the research. Finally, I would like to thank my family and friends.

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the author and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

Contents

List of Figures

List of Tables

Part I

Foundation

1 Optimization, Scheduling, and Search

This chapter introduces the problems of concern and describes the methods that are employed to solve them. First, problem concepts and solution techniques are described in general terms in Section 1.1 and 1.2. Next, the specific problems and solution techniques employed in this dissertaion are described in detail in Section 1.3 and 1.4. Section 1.5 concludes with a summary of the chapter.

1.1 Optimization Problems

Consider the following situation: You are the chef of a pizza restaurant. The restaurant is doing well and so at any given point in time there are a lot of pending pizza orders. Now your task as the chef is to arrange a sequence of the pizza orders such that all customers receive their pizzas in time. This is an example of a combinatorial optimization problem. Combinatorial optimization problems typically involve finding groupings, orderings, or assignments of a discrete, finite set of objects that satisfy certain conditions or constraints [Cook 97, Schr 03]. Combinations of these *solution components* form the potential solutions of a combinatorial problem. In case of the pizza scheduling problem above, it is clear that there exists a great many potential solutions in the form of sequences of pizzas. Each sequence is a *candidate solution*. The sequence that leads to the most timely delivery of the pizzas to the customers is the *optimal solution*.

It is useful to make a distinction between problems and instances. A *problem* is a description of a situation that needs to be resolved; an *instance* is a specific case of the situation described. Thus, in the pizza example, the problem is to arrange the pizzas in an optimal order, instances are the pizza orders and customers that the chef is faced with at a particular point in time. When several instances have common properties, they can be grouped in *instance classes*. For instance, a large group of customers of a fast food restaurant constitutes a different pizza problem instance than a small group of customers of a slow food restaurant. Several groups of slow food customers together form an instance class and so do several groups of fast food customers taken together. It is also possible to talk about problem classes. A *problem class* is the conjunction of two or more problems that have common properties. For example, arranging pizzas for one oven is one problem and arranging pizzas for multiple ovens is a different but related problem. Both problems belong to the more general class of pizza arrangement problems.

It was already mentioned that the space of candidate solutions for a combinatorial optimization problem is typically very large. In fact, it turns out that for many problems, the solution space is so large that it is infeasible to search all of it for any decent sized instances. The functional dependency between the size of an instance and the time and space required to solve this instance is known as the *complexity* of a problem [Gare 79]. Two particularly interesting complexity classes are $\mathcal{P}$, the class of problems that can be solved by a *deterministic* machine in polynomial time, and $\mathcal{NP}$, the class of problems which can be solved by a *nondeterministic* machine in polynomial time.[1] Every problem in $\mathcal{P}$ is also contained in $\mathcal{NP}$, as deterministic calculations can be emulated on a nondeterministic machine. However, the question whether the same is true the other way around and hence whether $\mathcal{P} = \mathcal{NP}$ remains a prominent open problem in computer science. A problem that is at least as hard as any problem in $\mathcal{NP}$ is called *NP-hard*. NP-hard problems may actually have a complexity that is higher than $\mathcal{NP}$. Those NP-hard problems that are contained in $\mathcal{NP}$ are called *NP-complete*. There are actually quite a number of problems that have proven to be NP–complete. In the worst case, the time that algorithms require to solve such problems grows exponentially with instance size. In addition, there are many problems that are NP–hard [Gare 79]. The $\mathcal{P} = \mathcal{NP}$ problem reduces to the question whether for any NP–complete problem, a polynomial algorithm can be found. On the grounds that many have tried and none succeeded, it is commonly believed nowadays that $\mathcal{P} \neq \mathcal{NP}$.

In the next chapter, we will see that the pizza scheduling problem can be NP-hard, NP-complete, or solvable in polynomial time, depending on details in the problem formulation: The problem of scheduling pizzas for a single oven is solvable in polynomial time when the total number of customers for whom the pizza arrives late is taken as measure; any complication to this basic sketch, be it multiple ovens, multiple stages in the baking process, or the assignment of different priorities to different customers, makes the problem NP-hard.

1.2 Approximate Algorithms

Given that many problems which we are interested in are NP–hard, there is little hope that we will be able to develop algorithms that are sure to find to optimal solution within the time we are prepared to wait. Luckily, for most purposes, it is sufficient to get a solution of a decent quality. Approximate algorithms are algorithms that find such solutions [Aart 97].

An *algorithm* is an abstract description of a computer program. Computer programs can be build out of subunits that have a single entrance point and a single exit. That is, the subunits can be regarded as machines that turn input into output. There are three ways to combine the subunits: *Sequencing* (do A, then B, then C), *alternation* (either do A, or do B) and *iteration* (repeat A until some condition

[1] Nondeterministic machines are hypothetical machines that can be thought of having the ability to make correct guesses for certain decisions.

is satisfied). Essentially all programs can be expressed with these three idioms [Bohm 66], but, for the description of approximation algorithms using the search metaphor may actually be more practical.

Search Paradigms

The fundamental idea behind the search approach is to iteratively generate and evaluate candidate solutions. In general, the evaluation of candidate solutions is problem dependent, whereas mechanisms for the generation of candidate solutions are applicable to a wide range of quite distinct problems. There are basically two mechanisms for solution generation: construction and perturbation. In the pizza scheduling problem above, the *construction* of a solution would be to take the pizza orders and to put them in a specific order. Here, the pizza orders are solution components and the order of pizza orders is the candidate solution. In contrast, the *perturbation* of a solution would be to take an established order of pizza orders and to modify the order. The *search space* of a problem instance is the collection of all solutions that can be generated through some generation mechanism. *Systematic search algorithms* traverse the search space of a problem instance in a systematic manner which guarantees that eventually either a (optimal) solution is found, or, if no solution exists, this fact is determined with certainty. *Local search algorithms*, on the other hand, start at some location of the given search space and subsequently move from the present location to a neighboring location in the search space, where each location has only a relatively small number of neighbors and each of the moves is determined by a decision based on local knowledge only. A final distinction among search approaches is between those that make use of randomized choices in generating or selecting candidate solutions for a given problem instance and those that do not. The former approach is *stochastic* search, the latter approach is *deterministic* search. Deterministic search will do the same each time it is applied to the same problem instance, stochastic search may perform differently.

Construction Heuristics Constructive search can be very weak or very powerful depending on the method of construction. Consider the pizza example again: A sequence of pizza orders can be constructed by *adding* a pizza order to a sequence until all pizza orders have been added. For this process to be successful, it is crucial that the pizza orders are picked in the right order. Typically, *dispatching rules* are employed to determine which of the remaining solution components has priority [Haup 89]. In the pizza example, one could decide to add the pizza that has the lowest baking time first — or one could decide to add the first pizza that is due. A sequence of higher quality can probably be constructed by *inserting* a pizza order on the best place in the current sequence. With insertion, the order with which one picks the pizza orders is probably less important than with addition. However, in contrast to addition, insertion also requires the evaluation of partial candidate solutions and the number of evaluations required increases as the construction proceeds. A third method next to insertion and addition is *backtracking*.

Backtracking is the ability to undo previous decisions. In the pizza example, backtracking would be to remove a pizza order from the sequence and add this order to the pool of pizza orders waiting to be put in sequence. Another pizza order from that pool can then be picked and put in place of the removed pizza order. With backtracking all feasible solutions can be constructed in one go and hence systematic search is possible.

We will see in Chapter 3 that construction on the basis of dispatching rules alone can yield quite good solutions after a sufficient amount of tuning. Yet, we will also see that construction by insertion tends to yield even better solutions — even without tuning.

Local Search Algorithms Local search [Aart 97] can be constructive as well as perturbative, deterministic as well as stochastic. In addition, local search algorithms distinguish themselves by the neighborhoods that they consider and by the strategy with which the search the neighborhoods. The *neighborhood* of a candidate solution is the set of alternate candidate solutions that are considered by the local search algorithm. For example, for the insertion construction algorithm described above, the neighborhood consists of all partial sequences of pizza orders that result from the insertion of the extra pizza order somewhere in the current sequence. Often, a neighborhood is defined by the type of *move* that the search algorithm will consider on the candidate solution. Constructive search on the pizza scheduling problem allows for insertion and addition. In perturbative search, one could consider *shifting* pizza orders a few positions in the overall order, or one could consider *swapping* pizza orders within the order — or any combination of both. The choice of the move is crucial as it determines the *size* of the neighborhood, the *ease* with which new candidate solutions can be evaluated, and the *topology* of the neighborhood structure. In turn, size, ease and topology are neighborhood characteristics that have to be considered in the choice of the strategy employed to search the neighborhood. Two common search strategies or *pivoting rules* are first improvement local search and best improvement local search. In *first improvement* local search, the algorithm scans the neighborhood of the current solution until a better solution is found. In *best improvement* local search, the algorithm scans the complete neighborhood and then selects the best candidate solution in that neighborhood as next solution. When the size of a neighborhood is small, the selection of pivoting rule will in most cases not have a great effect on the run time of the local search or the quality of the solutions that are found. For larger sized neighborhoods, the cost of scanning the whole neighborhood has to be balanced against the expected gain in solution quality. When candidate solutions can be evaluated easily, the cost of scanning the whole neighborhood will be limited. In cases where the landscape emerging from the combination of neighborhood structure with objective function is smooth, there is less need to scan the whole neighborhood than in cases where the neighborhood has a rugged topology.

Local search algorithms that search their neighborhoods according to the best

improvement or first improvement pivoting rule are also known as *iterative improvement* algorithms. With the shift and swap move and the first improvement and best improving pivoting rule as building blocks, one can define up to 14 varieties of iterative improvement. Chapter 4 describes how this is done and examines which variety induces the best performance. In addition to iterative improvement many variations on local search with more complex methods of scanning the neighborhood have emerged over the years. Of them, simulated annealing [Kirk 83] and tabu search [Glov 89, Glov 90] are the most well known. However, in the context of this dissertation, we will ignore these methods in favor of the more general extensions introduced in the next section.

Search Extensions

The local search paradigm can be extended in several ways. In order to describe these extensions, often additional metaphors are introduced. Examples of such metaphors are annealing [Kirk 83], evolution [Gold 89], and ant foraging behavior [Dori 99].[2] Here, I want to stick to the more profane level of description introduced on page 4: The description of the extensions in terms of sequencing, alternation and iteration.

Local search algorithms and construction heuristics can be used as subunits, modules, or components in superstructures known as metaheuristics [Osma 96]. For instance, most metaheuristics employ a construction heuristic to construct the initial solution and then employ perturbative local search to find a better solution. Alternatively or in addition to this, a metaheuristic could employ different search methods depending on characteristics of the problem instance it needs to solve. Moreover, a metaheuristic could repeat the same method or set of methods over and over again.

Piped Local Search Sequencing of search components has for many years been the research topic of Hansen and Mladenović [Hans 03, Hans 99a, Mlad 97]. This research has resulted in a variety of algorithms and among them variable neighborhood descent is the most elegant.[3] *Variable neighborhood descent* exploits the fact that a local optimum with respect to one neighborhood structure is not by necessity also a local optimum for another neighborhood structure. That is, when a local search algorithm cannot find any improving solutions in one neighborhood anymore, another neighborhood may still contain improving solutions. Moreover, a global optimum is a local optimum with respect to all possible neighborhood structures and for many problems local optima with respect to one or several neighborhoods are relatively close to each other and so the likelihood of being close to a global optimum is higher, if a solution is a local optimum with respect to many neighborhoods. Therefore, one can expect that a local optimum provides

[2] cf. [Brad 85, Coll 88, Bona 99, Dori 04].

[3] Other varieties of variable neighborhood search (VNS) are, among others, variable neighborhood decomposition search, skewed VNS, and reduced VNS [Hans 03].

information about a global optimum and that hopping from one optimum to another one brings the algorithm closer to the global optimum.

Variable neighborhood descent searches through a series of neighborhoods in a predetermined order. If a neighborhood does not yield any improvement any more, the next neighborhood is considered. If a neigbhorhood does yield an improvement, this improvement is applied and the search for further improvements is started again in the first neighborhood of the series. For the implementation of variable neigbhoorhood descent, it is up to the developer to decide what neighborhoods should be used and how many of them. In addition, he or she should decide in which order the neighborhoods are searched. Hansen and Mladenović offer some suggestions as to what might work [Hans 99b]. Nevertheless, it is advisable to validate these choices experimentally.

In Section 4.2, the implementation of a close kin to variable neigbhorhood descent is discussed. This kin is coined *piped iterative improvement* (PII). Like variable neighborhood descent, PII searches through a series of neighborhoods in a predetermined order. Unlike variable neigbhorhood descent, PII only returns from the current neighborhood to the first neighborhood if it is absolutely certain that no more improvement can be found in the current neighborhood. PII is an instance of piped local search [Best 01a]. In *piped local search*, each local search is approached as a black boxes where initial solutions are put in and improved solutions are returned. The concatenation of the local searches is what produces the piped local search. The idea of piping is more general than that of varying neighborhoods in that piping allows its constituents to differ in more aspects than neighborhood alone. The *shifting* approach [Barb 00], for instance, pipes procedures with distinct problem representations. Yet, PII in Section 4.2 limits itself to piping of local searches that only differ with repect to the neighborhood that is used.

Iterated Local Search Iteration of local search has been researched in depth by Lourenço, Martin, and Stützle [Lour 02]. This research resulted in the development of a framework known as iterated local search (ILS). Iterated local search has three components: local search, perturbation, and an acceptance criterion. If a starting solution is not already available, a fourth component, the construction heuristic, also has to be configured. *Local search* takes an initial solution and turns it into a better solution. The *perturbation* then modifies the current solution such that it is no longer a local optimum for the local search. Local search is applied again and the *acceptance criterion* is used to determine whether the new solution should be accepted in favor of the old solution or not. The perturbation–search–acceptance cycle continues until the user deems the incumbent solution to be good enough for his or her purposes.

ILS exploits the stylized fact that the quality of initial solutions for local search tends to be positively related to the quality of the ultimate solutions the local search is able to find. Thus, a local search that starts near a local optimum is more likely to find a better local optimum that a local search that starts from a random

point in the search space. The implementation of ILS involves the selection of local search, the configuration of a perturbation mechanism, and the specification of an acceptance criterion. Like their colleagues did for variable neighborhood descent, Lourenço, Martin, and Stützle offer suggestions as to what might work. Nevertheless, it is again advisable to validate these choices experimentally (which is what we do in Chapter 5).

1.3 Scheduling Problems

Scheduling problems are a special class of combinatorial optimization problems concerned with the allocation of tasks to resources [Pine 95, Ande 97, Conw 67, Grah 79]. In the context of scheduling, it is common to call tasks *jobs* and resources *machines*. There exists an agreed upon vocabulary to describe these problems. Associated with the vocubulary is a shorthand. Adherence to vocabulary and shorthand has the advantage that it allows for an easy classification of problems. After an introduction of the vocabulary and the shorthand, this section then shows how the shorthand can be used to deduce the relationships between problems. Finally, this section describes the subset of scheduling problems attacked in this dissertation.

Terminology

A large number of scheduling problems and models have been studied and analyzed in the literature. See [Pine 95] for a good overview.

Scheduling models can be classified as either deterministic or stochastic. *Deterministic* models suppose that the job data are exactly known in advance. In *stochastic* models, on the other hand, only the distribution of the data is known. The actual processing times, release dates and due dates are known only after the completion of the processing or after the actual occurrence of the release or due date.

Here, we focus on deterministic models. The large amount of research put into these models during the last four decades shows that they are already complex enough to be of strong interest. Besides, the static nature of deterministic models is not as restrictive as it may look on first sight, since many dynamic problems can actually be solved as a series of static problems.[4]

Job characteristics The following pieces of data are associated with job j:

Processing time (p_{ij}): p_{ij} represents the processing time of job j on machine i. The subscript i will be omitted in the following when the problems involve only one machine or when processing times on other machines are the same as the processing time on the first machine.

[4] Problems whose topology and costs do not change while they are being solved are called *static*; problems whose topology and costs do change are called *dynamic*.

Release date (r_j): The release date r_j of job j is the time the job arrives at the system, that is, the earliest time at which job j can start its processing.

Due date (d_j): The due date d_j of j represents the committed shipping or completion date, the date the job is promised to the customer. The completion of a job after its due date is allowed, but a penalty is incurred. When the due date must absolutely be met, it is referred to as a deadline.

Weight (w_j): The weight w_j of job j is basically a priority factor. It denotes the importance of job j relative to the other jobs in the system. For example, a weight may represent the actual cost of keeping the job in the system.

The problems dealt with in this dissertation assume that all jobs are immediately available for processing, that setup times are sequence independent and added to the processing times, and that jobs are not subject to precedence constraints. Furthermore, the processing of a job cannot be interrupted and the job has to be kept on the machine until completion, *preemption* is not allowed.

The distribution of the problem data affects the difficulty of a problem instance. For instance, if all jobs have the same processing time, release date, weight, and due date, then the ordering of the jobs does not affect any common performance measure. The more the job data differ, the more difficult it is likely to become to find the best ordering of jobs vis-a-vis some objective.

Indicators are often used to characterize distributions of job data. Most common are tardiness factor (TF) and range of due dates (RDD) which indicate how harsh the due dates for the jobs are. TF compares the first moment of the due date distribution, its mean, to the makespan $C_{\max}$, the minimum total processing time, of the jobs in the system.

$$TF = 1 - \frac{\overline{d}}{C_{\max}} \tag{1.1}$$

RDD compares the spread of the due dates, given by subtracting minimum from maximum due date, against the makespan.

$$RDD = \frac{d_{\max} - d_{\min}}{C_{\max}} \tag{1.2}$$

Machine environments Different configurations of machines are possible. For our purposes, machines are assumed to be available to process jobs at time zero without interruption.

Some of the better known machine environments are defined as follows:

Single Machine (1): There is one machine.

Parallel Machines (Pm): There are m homogeneous machines in parallel. That is, job characteristics are the same regardless on which machine the jobs are processed. Each job has to be processed on one of the m machines.

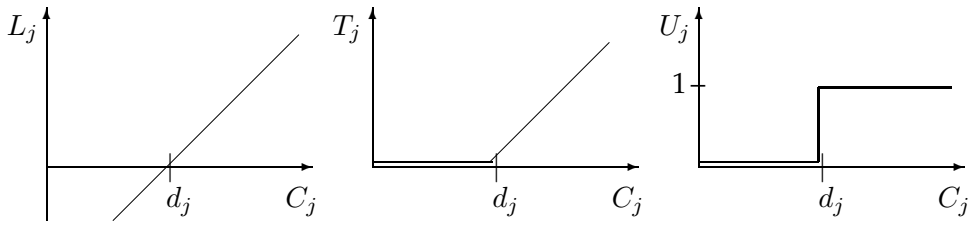


Figure 1.1: Illustration of due–date related penalty functions adopted from [Pine 95, Fig. 2.1]. Let d_j and C_j be the due date and completion time of job j. Then L_j, T_j, and U_j are the lateness, tardiness and unit penalty of this job, respectively.

Flow Shop (Fm): There are m machines in series. Each job has to be processed on each of the m machines. All jobs have the same routing, that is, they have to be processed first on machine 1, then on machine 2, and so on. After completion on one machine, a job joins the queue at the next machine. Usually, all queues are assumed to operate under the *first in first out* (FIFO) discipline, that is, a job cannot "pass" another while waiting in a queue. If the FIFO discipline is in effect, the flow shop is referred to as a *permutation* flow shop. This feature is often abbreviates as *prmu*.

Job Shop (Jm): In a job shop with m machines, each job has its own route to follow. A distinction is made between job shops where each job visits any machine at most once and job shops where a job may visit a machine more than once. In the latter case, the β field contains the entry *recrc* for *recirculation*.

In addition to these machine environments, one can also encounter uniform parallel machines, unrelated parallel machines or open shops — that is environments with machines with different speeds, environments with machines with different speeds dependent on the jobs, and environments where jobs have to be processed on each machine be it sometimes with zero processing time — but these environments are less prevalent.[5]

Objectives The objective to be minimized is always a function of the completion times of the jobs. The *completion time* of job j — that is, the time at which it exits the system — depends on the schedule and is denoted by C_j. In addition, many objectives take due dates into account. Figure 1.1 illustrates the three most common penalty functions related to due dates. The *lateness* of job j is defined as $L_j = C_j - d_j$. It is positive if job j is completed late and negative if it is completed early. The *tardiness* of job j is defined as $T_j = \max\{C_j - d_j, 0\} = \max\{L_j, 0\}$. The difference between tardiness and lateness lies in the fact that tardiness is never negative.

[5]Still, there are some applications. See [Lens 90, Lawl 79].

Complementary to tardiness one can encounter *earliness*, $E_j = \max\{d_j - C_j, 0\}$. Finally, *unit penalty* $U_j = 1$ can be incurred for a job j if $C_j > d_j$.

Typically, the objective of a scheduling problems is to minimize the maximum or the sum of the penalties. Here are some examples.

Makespan ($C_{\max}$): The *makespan*, defined as $\max(C_1, \ldots, C_n)$, is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a high utilization of the machine(s).

Maximum lateness ($L_{\max}$): The maximum lateness, $L_{\max} \equiv \max(L_1, \ldots, L_n)$, measures the worst violation of the due dates.

Total weighted completion time ($\sum w_j C_j$): The sum of the weighted completion times of jobs gives an indication of the total holding, or inventory, costs incurred by the schedule. The sum of the completion times is in the literature often referred to as the *flow time*. The total weighted completion time is then referred to as the *weighted flow time*.

Total weighted tardiness ($\sum w_j T_j$): The weighted sum of the tardiness of jobs gives an indication of the extent with which jobs miss their target delivery date.

Weighted number of tardy jobs ($\sum w_j U_j$): The weighted number of tardy jobs is not only a measure of academic interest, it is often an objective in practice as it is a measure that can be recorded very easily.

Three–Field Representation It is convenient to adopt the representation scheme of [Grah 79], cf. [Ande 97]. This is a three–field descriptor $\alpha|\beta|\gamma$ which indicates problem type: α represents the machine environment, β defines the job characteristics and γ is the optimality criterion.

Let $\circ$ denote the absense of a symbol. The first field takes the form $\alpha = \alpha_1\alpha_2$, where α_1 and α_2 are interpreted as follows:

- $\alpha_1 \in \{\circ, P, Q, R, F, O, J\}$
  - $\alpha_1 = 1$: a single machine
  - $\alpha_1 = P$: identical parallel machines
  - $\alpha_1 = Q$: uniform parallel machines
  - $\alpha_1 = R$: unrelated parallel machines
  - $\alpha_1 = F$: a flow shop
  - $\alpha_1 = O$: an open shop
  - $\alpha_1 = J$: a job shop
- $\alpha_2 \in \{\circ, m\}$

- $\alpha_2 = \circ$: the number of machines is arbitrary
- $\alpha_2 = m$: there are a fixed number of machines m

Note that for a single machine problem $\alpha_1 = \circ$ and $\alpha_2 = 1$.

The second field β indicates job characteristics. β can be quite complex since there are many characteristics that can be set. It goes too far to set out all intricacies here. For the purpose of this dissertation, jobs have a processing time, weight, and due date associated with them; they have neither release dates nor setup times; and there are no precedence constraints. Therefore, with the exception of flow shop problems, β remains empty and the three–field descriptor takes the form $\alpha||\gamma$. In case of the flow shop problems $\beta = prmu$ to indicate that we are concerned with *permutation* flow shop problems.

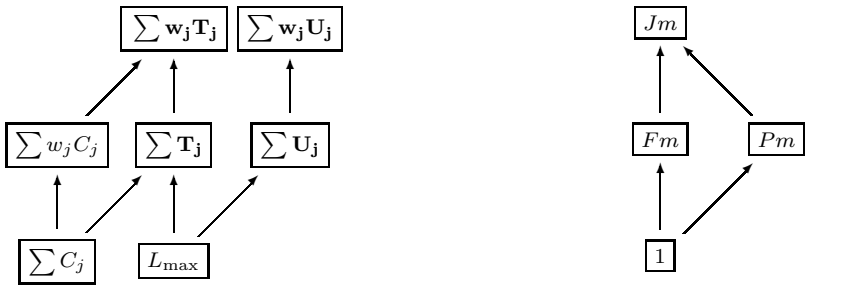
The third field defines the optimality criterion, which involves the minimization of $\gamma \in \{C_{\max}, L_{\max}, \sum(w_j)C_j, \sum(w_j)T_j, \sum(w_j)U_j, \sum(w_j)E_j\}$, to list the most common criteria. Furthermore, it is sometimes appropriate to adopt a composite objective, one component of which may be a setup cost.

Hierarchy

Often, an algorithm for one scheduling problem can be applied to other scheduling problems. For example $1||\sum C_j$ is a special case of $1||\sum w_jC_j$, and a procedure for $1||\sum w_jC_j$ can of course, be used also for $1||\sum C_j$. In complexity terminology it is then said that $1||\sum C_j$ reduces to $1||\sum w_jC_j$. This is usually denoted by $1||\sum C_j \propto 1||\sum w_jC_j$. Based on this concept, a chain of reductions can be established. For example, $1||\sum C_j \propto 1||\sum w_jC_j \propto Pm||\sum w_jC_j$. Of course, there are also many problems that are not comparable with one another: e.g. $Pm||\sum w_jT_j$ and $Jm||C_{\max}$.

A considerable amount of effort has been dedicated to the establishment of a problem hierarchy that describes the relationships between the scheduling problems known in the literature. In the comparisons between the complexities of the different scheduling problems, it is of interest to know how a change in a single element in the classification of a problem affects its complexity. Most of the hierarchy is relatively straightforward. Figure 1.2 on the next page gives the complexity relationships for some objective functions and some machine environments. Arrows point from a special case to a more complex generalization.

Like in combinatorial optimization in general, a significant amount of research in deterministic scheduling has been devoted to finding efficient, polynomial time, algorithms for scheduling problems. However, again like in combinatorial optimization, many scheduling problems do not have a polynomial time algorithm and are *NP–hard* [Lens 97, Lawl 93, Bruc 03]. Research in the past has focused in particular on the borderline between polynomial time solvable problems and NP–hard problems. For example, in the string of problems described above, $1||\sum w_jC_j$ can be solved in polynomial time, but $Pm||\sum w_jC_j$ is NP–hard.



(a) Objective Functions

(b) Machine Environments

Figure 1.2: Complexity hierarchies based on problem characteristics.

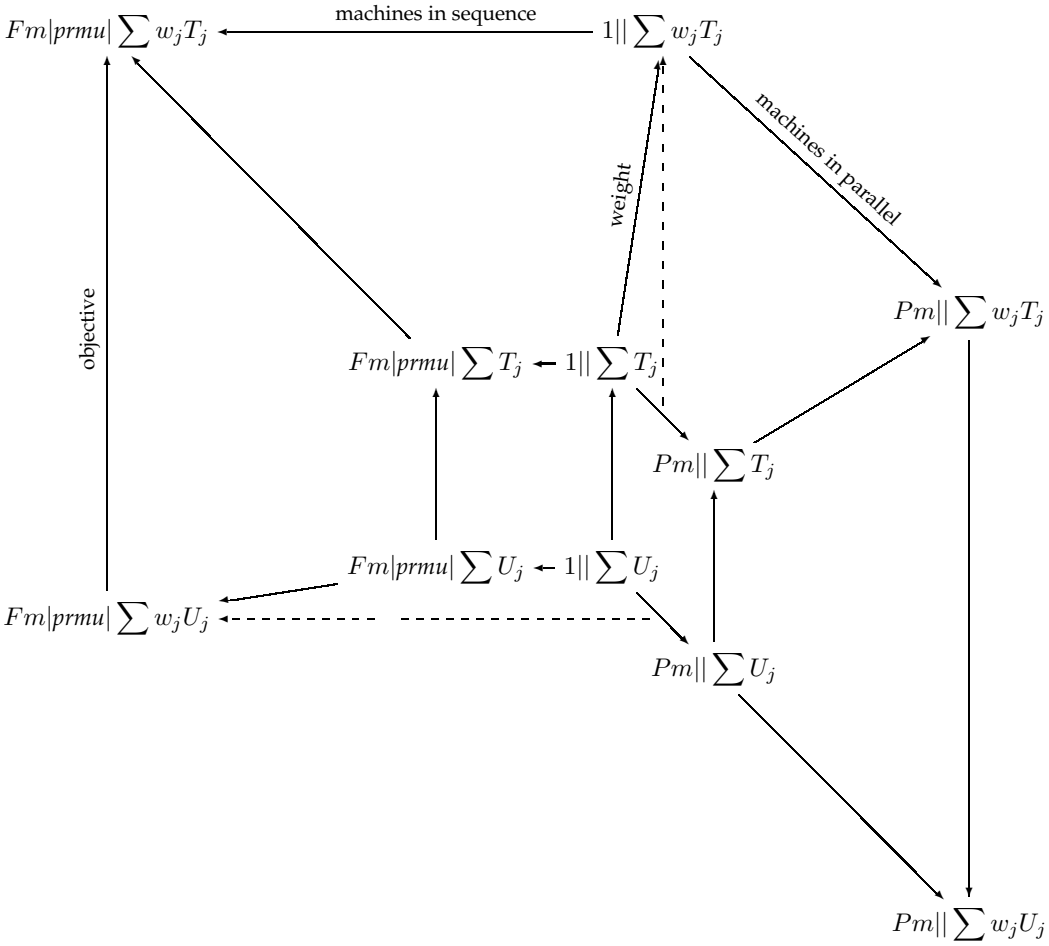


Figure 1.3: Hypercube mapping of selected scheduling problems.

Selection
At the beginning of this chapter, an example of an optimization problem was given where the goal is to devise an order in which pizzas should be put in an oven in order to minimize the time that pizza customers have to wait. In fact, the problems of concern in this dissertation can be seen as instantiations of this example.

Problems Figure 1.3 on the facing page depicts this selection of scheduling problems. The figure is a partial hypercube where the problems fall apart in four dimensions: the number of machines in sequence (x-axis), the number of machines in parallel (y-axis), the shape of the due date penalty (z-axis), and whether or not jobs have weights (w-axis). Apart from $1||\sum U_j$ all problems are NP-hard.[6]

In the pizza analogue, $1||\sum U_j$ is the problem of scheduling pizzas for a single oven, where each pizza has a specific baking time and the unit penalty consists of the loss of a customer whenever a pizza is late. When customers do not walk away immediately, but instead grow more angry over time, the corresponding problem is $1||\sum T_j$. When some customers are more important than others, for example because they pay more, the corresponding problem is $1||\sum w_jU_j$ or $1||\sum w_jT_j$. The descriptor Pm is in place if more than one pizza fits in the oven or where there are multiple ovens. Finally, a pizza flow shop problem occurs when the chef is not only interested in baking pizzas but also wants to take earlier stages in the preparation — chopping tomatoes, kneading dough — into account. Each stage is a machine and the number of stages is indicated by the value of m in the descriptor Fm.

Variations in machine environment and weights are chosen so as to be able to investigate the effect these variations have on algorithm performance. Due-date related objectives are chosen because they seem underrepresented relative to makespan in academic research [Beck 97]. Moreover, even an objective like $\sum U_j$, which at first might appear somewhat artificial, can be of great practical interest. Equivalent to the percentage of on-time shipments, it is easy to monitor and therefore often used to rate managers' performance [Pine 95, §3.3]. Also weights can be easily justified. For, job priority can be interpreted as reflecting a strategic weight that is attached to the costs of tardy deliveries. These costs, be it customer badwill, lost future sales, or rush shipping costs may vary significantly over customers and orders and that is why weights may be appropriate [Veps 87].

There are twelve distinct problems. These twelve problems are subdivided into twenty four instance classes: There are four classes with a single machine environment for instances with 100 jobs ($1||\sum T_j$, $1||\sum U_j$, $1||\sum w_jT_j$, $1||\sum w_jU_j$). There

[6] For $1||\sum U_j$, there exists an algorithm that can solve its instances in polynomial time ($O(n \log n)$) [Moor 68]. For $1||\sum T_j$ and $1||\sum w_jU_j$, there exist algorithms that require pseudo–polynomial time ($O(n^4 \sum p_j)$, to be precise) [Lawl 77, Lawl 69]. Yet, both problems are known to be NP–hard [Lawl 69, Karp 72, Du 90]. The problems $Pm||\sum U_j$ and $Fm|\text{prmu}|\sum U_j$ are also NP–hard, since $Pm||C_{\max}$ is NP-hard [Gare 78] and $F3|\text{prmu}|C_{\max}$, $F2|\text{prmu}|L_{\max}$ are NP-hard [Gonz 78, Cho 81]. All the other problems are NP–hard since they are extensions to either $1||\sum T_j$, $1||\sum w_jU_j$, $Pm||\sum U_j$, or $Fm|prmu|\sum U_j$.

are four problem classes with 100 jobs and two machines in parallel and also four with three and four machines in parallel. Finally, there are four classes of flow shop problems with 50 jobs and 20 machines and four classes of flow shop problems with 100 jobs and 20 machines.

Test Instances Instances in the single and parallel machine instance classes consist of jobs with processing times uniformly distributed between 1 and 100, weights uniformly distributed between 1 and 10, and due dates distributed in such a way that in the aggregate the tardiness factors (TF) and ranges of due dates (RDD) are uniformly distributed between 0 and 1. That is, for an individual instance, the due dates are uniformly distributed between $(1 - \mathrm{TF} - \frac{\mathrm{RDD}}{2}) \cdot C_{\max}$ and $(1 - \mathrm{TF} + \frac{\mathrm{RDD}}{2}) \cdot C_{\max}$, where TF and RDD are random variables. In the single machine environment, $C_{\max}$ is the sum of all processing times. In the parallel machine environment $C_{\max}$ is approximated by dividing the sum of the processing time by the number of machines. For flow shop instances the processing times are taken from one of the ta51-60 and ta81-90 benchmark instances [Tail 93]. For these instances $C_{\max}$ is known. Or, more precisely, very good upper bounds are available after years of attempts to solve these instances. Due dates and weights are then drawn from the same distributions as due dates and weights in the single machine and parallel machine cases.

This particular selection of problem instances is made with the hope that it covers all sources of variation within even the subset of scheduling problems that are of importance. It seems safe to assume that this hope is well–founded since the benchmark instances on which the fully developed algorithms will ultimately be tested have been generated along similar lines (see Chapter 6). However, this still leaves the question pending whether or not the benchmark instances themselves are of any importance to the outside world.

Caveat Here is a little note of warning: The proposed scheduling problem instances are highly artificial and so conclusions drawn from the instances are more likely than not applicable to industrial scheduling problems. With regards to objective functions note that in the real world one often tries to minimize multiple objectives at the same time. Plus, in practice, the due date function is probably more like a sigmoid. That is, it is not piecewise linear but lies somewhere in between the tardiness function and the unit penalty function. With regards to processing times, note that distributions are often more structured. There may be a correlation between processing times and machines and processing times may change due to learning or deterioration (Granted, this is a rather long–term effect). Weights of jobs often fluctuate over time. Finally, the real world is less static than is assumed. In the real world, there may be n jobs in the system at any time, but new jobs are added continuously. Moreover, there is no perfect knowledge of the future. Last but not least, processing restrictions and constraints may be very involved. However, given the tremendous difficulty that is associated with finding

Table 1.1: Example instance of a 10 job single machine problem.

job id	A	B	C	D	E	F	G	H	I	J
processing time	54	21	49	52	19	10	20	41	58	52
weight	9	3	3	2	6	8	7	7	3	9
due date	31	64	14	242	305	176	64	249	353	376

real world problems and the ease with which new instances of artificial problems can be generated, the selection and use of problem instances carried out in this dissertation still is very commendable.

1.4 Solving Scheduling Problems

The scheduling problems on which we focus here have one important commonality: The solutions for the problems can easily be represented as a permutation of jobs. The permutation representation is not only natural, it has also been found to work well — at least for $1||\sum T_j$ [Crau 98].

As was explained in Section 1.2, there are basically two things one can do with the permutation representation. One can either construct a permutation or one can permute a permutation. Permutation construction consists of consecutively adding jobs to the front or back of the permutation or inserting the jobs somewhere in between. Permutation of permutations can be done either by swapping two jobs or by shifting a series of jobs backwards or forwards. More specifically, swap and shift are carried out as follows: Let (i, j) be a pair of positions and $\pi = (\pi_1, \ldots, \pi_i, \ldots, \pi_j \ldots, \pi_n)$ the current permutation. A *swap* of i and j yields $\pi' = (\pi_1, \ldots \pi_j, \ldots, \pi_i, \ldots, \pi_n)$; a *shift backwards* of yields $\pi' = (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_j, \pi_i, \pi_{j+1}, \ldots, \pi_n)$; and a *shift forward* of yields $\pi' = (\pi_1, \ldots, \pi_{j-1}, \pi_i, \pi_j, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n)$. Swap, forward shift, and backward shift are illustrated in Figure 1.4 on page 19.

A practical difference between algorithms based on construction and those based on permutation is that the latter involves a large number of evaluations of intermediate solutions whereas the former involves of small number of partial solutions. Therefore, achieving efficiency of evaluation is much more important in the context of permutation. And that is why the focus will be on that aspect in this section.

The evaluation of the scheduling solutions is different for each of the twelve problems. Self–evidently, the inclusion of weights and the shape of the penalty function affects the evaluation. However, the most profound impact on problem solution implementation is due to the choice of machine environment.

Single Machine Environment

Table 1.1 contains the job data for an example single machine problem instance. The instance has ten jobs, one job per table column and each job has a processing time, a weight, and a due date associated to it. A feasible solution to this instance is to put the jobs in alphabetical order. This is not the optimal solution since the

solution has 6 late jobs with a total weight of 37, a total tardiness of 351, and a total weighted tardiness of 2048. Alternatively, if you put the jobs in increasing due date order, there would be 4 late jobs with a total weight of 22, a total tardiness of 247, and a total weighted tardiness of 1493. The latter is clearly better.

Figure 1.4 on the next page shows Gantt charts for an initial solution of jobs in alphabetic order and several permutations to this solution. In a *Gantt chart*, a job is represented by a box whose width is determined by the job's processing time. The arrangement of the boxes represents the order of the jobs' execution over time. The job on the left is executed first and the job on the right is executed last. When there are multiple machines, the chart shows multiple sequences of jobs — one sequence per machine.

The first thing to note is that a move involving positions i and j only affects the completion times of jobs between i and j and so one only has to consider those jobs in order to get to know the effect of the move. The second thing to note is that a shift is equivalent to a swap where one of the jobs involved in the swap has a processing time of zero.

The effect of a swap of jobs on positions i and j depends on the difference between the processing times of the jobs. If the processing time of job j is smaller than the processing time of job i then the jobs between i and j will start earlier after the swap. Jobs that were not late, will not become late and jobs that are late become less late. In contrast, if the processing time of job j is larger than the processing time of job i, then jobs between i and j will start later after the swap. Jobs that were already late will become more late and jobs that were not late might become late. In order to compute the result of a swap efficiently, one can split the permutation into runs of late and non-late jobs. If one keeps track of completion times and partial sums of tardiness in addition to this, one can easily and cheaply compute the effect of the swap.

Details on the way to efficiently evaluate the effect of swap moves together with details on the way to limit the number of moves are evaluated in the swap neighborhood without loss of quality are described for $1||\sum w_j T_j$ in [Cong 02]. The local search algorithms that were developed and tested for this dissertation adopt these details in case of the swap neighborhood and adapt them in case of the other neighborhoods. Moveover, the speed–ups are carried over to the other objective functions as much as possible. $1||\sum T_j$ employs the same implementation as $1||\sum w_j T_j$ and in case of $1||\sum(w_j)U_j$ the implementation is adapted to reflect the fact that unit penalties rather than tardiness penalties are summed. That is, data structure recording partial total tardiness are left out and estimations of potential improvements are simplified. Readers interested in the exact details are invited to peruse the implementation source code which is available upon request.

Parallel Machine Environment

The parallel machine environment consists either of multiple identical machines or of one machine that can process multiple jobs are the same time. Permutations alone cannot serve as solution representations for parallel machine problems, since

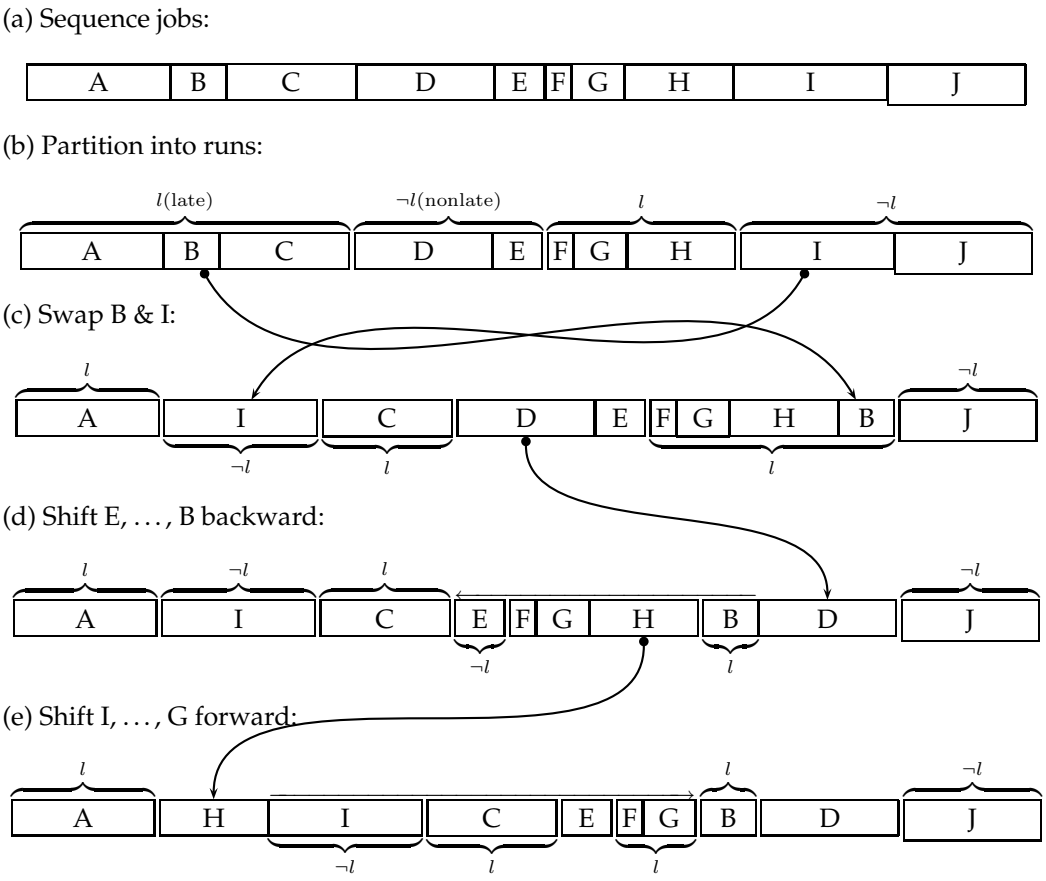


Figure 1.4: Evaluation of tardiness in a job schedule for a single machine.

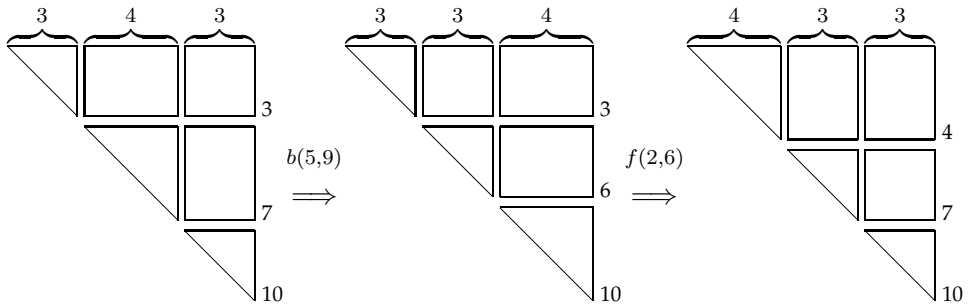


Figure 1.5: Mapping of moves from parallel machines to a single machine: Three examples.

they only convey job order not machine assignment. In the implementation for this dissertation, three methods are used to determine the assignment of jobs to machines.

The first method, greedy assignment, is used for the construction of a solution based on orderings of jobs sorted on the basis dispatching rules. *Greedy assignment* goes through all jobs one after the other and assigns each jobs to the first machine that becomes available for processing. That is, for a given permutation of jobs, the first m jobs are assigned to machine 1 to m and, one by one, the other jobs are assigned to the machine whose jobs have the smallest total processing time so far.

The second method is actually nothing more than a mere convention to make easy interaction between algorithmic components possible. In the single machine and in the flow shop environments a solution is just an array of numbers where each number identifies a job. In the parallel machine environment the solution is still represented by an array of numbers but this time around the number indicates job identity as well as machine allocation. The way this feat is achieved is simply by incrementing the number identifying the job with $k \cdot n$ if the job is to be assigned to the k^{th} machine and n is the number of jobs. Say, the number is i, then $i\%n$ reveals the job identity and i/n reveals to with machine the job is allocated.

The third method is the most intricate and is used within local search. Its purpose is to map a neighborhood scan on a single permutation of jobs to an internal representation of multiple permutations — one for each machine. The advantage of sticking to a single permutation representation is that in this way we can use the same operators and neighborhood that are also used in the single machine and flow shop environments.

Figure 1.5 shows how moves on the permutations c, d, and e in Figure 1.4 on the preceding page would be mapped to moves on corresponding permutations on multiple machines. The figure consists of three sets of blocks and triangle. Each set corresponds to one permutation. Blocks represent moves within a machine and triangles represent moves between machines in the multiple permutation representation. Now, moves on a single permutation involve two positions i and j. The x-axis maps the first position to a triangle or block and the y-axis maps the second position to a triangle or block. Hence we have to read leftmost set of blocks and triangles as follows: There are 10 positions on the single permutation representation; the first 3 positions correspond to moves on machine 1, the next 4 positions correspond to moves on machine 2 and the last 3 positions correspond to moves on machine 3; and a move involving one of the first 4 positions and one of the last 3 positions corresponds to a move between machine 1 and machine 3.

Even though the size of the single permutation is fixed, the mapping to the underlying multiplicity of permutations cannot be hard wired. Recall that the sets of triangles and blocks in Figure 1.5 represent the mappings of single permutations c, d, and e in Figure 1.4 to a three separate permutations. In Figure 1.4 the difference between c and d is a shift of job D from position 4 to position 9. This corresponds to the removal of the first job from permutation of the second machine and the insertion of this job on the forelast position in the permutation of

the last machine. Thus, the shift results in a smaller permutation for machine one and a bigger permutation for machine three. Also the transition from permutation d to e affect the underlying mapping. This time around, the shift of job H from position 7 to position 2 corresponds to a removal of the job from the first position on the permutation of the third machine and the insertion of the job on the second position of the permutation of the first machine.

Being forced to update the mapping is one of the reasons why it might pay off to introduce ways to discourage the algorithm from considering moves that involve a move of jobs between two machines. Another reason is that one can adopt the efficient single machine evaluation procedures for moves within machines. Between machines one cannot. The simplest way to guide the algorithm is to present it with the cheap moves first. That is, enumerate all pairs of indices and in the order in which they are scanned by the algorithm and map those pairs first to internal move pairs and next to external move pairs. In that way, at least a first improvement local search algorithm will not consider a move between machines if it can find improvements in other ways.

Flow Shop Environment

The single permutation representation is perfectly valid for the permutation flow shop environment and does not need any additional tampering. However, the relationship between a job's position in the permutation and its completion time is less clear cut. To see why, consider the Gantt chart of a ten job sequence in a flow shop of 8 machines in Figure 1.6 on the next page. The completion time of a job is the time at which the job exits the last machine. The relative order of the jobs on the last machine is the same as on the first machine. Nonetheless, one cannot simply sum the processing times of the jobs on the last machine because the release date of the jobs on that machine not only depends on the completion time of the previous job but also on the completion time of the job itself on the previous machine.

Computing the completion time of operations and/or jobs in permutation flow shop problems is done as follows:

Let $C_{i,j}$ be the completion time of the j^{th} job in the permutation on machine i, $p_{i,j}$ be the processing time of this job on this machine. The first machine behaves like a single machine:

$$C_{1,j} = \sum_{k=1}^{j} p_{1,k} \tag{1.3}$$

The first job can start on the next machine as soon as it has exited the current machine:

$$C_{i,1} = \sum_{k=1}^{i} p_{k,1} \tag{1.4}$$

The other jobs can start on the next machine as soon as they have exited the current machine and their predecessor is not occupying the next machine any more.

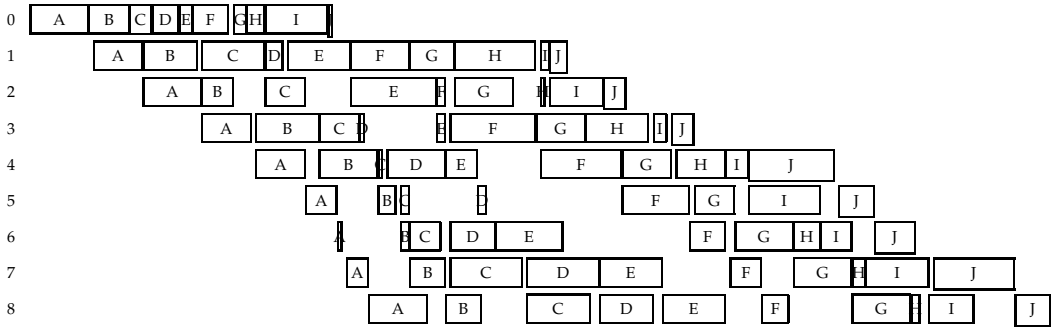


Figure 1.6: Gantt chart for a permutation flow shop solution.

$$C_{i,j} = \max\{C_{i-1,j}, C_{i,j-1}\} + p_{i,j} \tag{1.5}$$

In case of tardiness problems, we are only interested in the completion times on the last machine — but these depend on completion times of the preceding machines.

Just like in the single machine case, a move involving jobs on position i and j has no effect on the tardiness of jobs before $\min\{i, j\}$. Unlike the single machine case, in the flow shop case, the move can also affect the tardiness of jobs after $\max\{i, j\}$. The completion time remains the same on the first machine. On the other machines, the move may result in an increase of the time that the machines stay idle (represented by a gap in Gantt chart) and hence an increase of the completion times after the machine resumes its work. If the completion time for a job $k : k > \max\{i, j\}$ on the new schedule is the same as on the previous schedule for each machine, then the tardiness for jobs $k, \ldots, n$ is not changed. In the implementation, the above condition is checked and the updating of completion time and tardiness values is halted as soon as the condition is passed.

The makespan of a flow shop schedule only depends on the critical path of the last job on the last machine. Many perturbations of the schedule do not alter the critical path and so they need not be considered [Nowi 96]. Total tardiness, however, depends on the critical paths of each job on the last machine. So, hardly any move can be excluded on that basis. Hence, these speed–ups do not work in our case.

1.5 Summary

Combinatorial problems arise in many areas of computer science and other disciplines in which computational methods are applied. Several methods have been proposed to try and solve combinatorial problems. In this dissertation, problems solving is done by iteration and sequencing of local search.

Scheduling, the allocation of tasks to resources, is an activity which has been actively researched for over a century. Scheduling problems can be described in terms of machine environment, job characteristics and objective to be minimized. In this thesis, we focus on problems where regular jobs that are all available for scheduling from the start have to be scheduled on a single machine, parallel machine or flow shop machine environment such that the total tardiness of the jobs is minimized. Jobs can have priorities assigned to them and the penalty for being tardy can be a constant or increase with the amount of tardiness of the job. Solutions for these scheduling problems can easily be represented as a sequence of jobs. In the single and flow shop machine environments, the sequence equals the order of jobs that enter the (first) machine. In the parallel machine case, the sequence first has to be divided into subsequences. Within a sequence you can either swap or shift jobs. For the single machine environment a host of efficiency improvements can be attained in the computation of the tardiness. These improvements carry over to the parallel machine case in so far as the problem can be split up into several parallel single machine problems. However, in case of the flow shop environment, it seems to be a lot harder to obtain similar efficiency gains.

2 Calibration, Tuning, and Analysis

This chapter describes the method for algorithm development that is adopted for this dissertation. Algorithm development is done in an empirical manner. That is, components are added or removed and parameters are set on the basis of systematic and thorough testing. Furthermore, the test results are examined carefully so as to be able to find out why certain configurations work better than others. Last but not least, the process of developement is repeated on different problems so that the effect of problem specification on algorithm configuration can be assessed.

Systematic reporting on algorithm development is still rare in combinatorial optimization research. Yet, the need for such reporting is widely understood.[1] The prevailing practice is criticised [Hook 94, Hook 96] and guidelines have been written for those who intend to better their lives [Cohe 95, Barr 96]. Still, few researchers attain the level of rigour common in other empirical fields [Mont 91]. The methods proposed in this chapter are another step in that direction.

Section 2.1 details the procedure that is followed to enumerate and sift through algorithmic candidates. Next, Section 2.2 discusses how algorithm performance is to be measured and Section 2.3 describes the tools that are employed for the analysis of the experimental results. Finally, a summary is provided at the end of the chapter.

2.1 Candidate Selection through Racing

Racing is the preferred method for candidate selection in this dissertation. First, it is made clear why a procedure for candidate selection is needed. Next, racing is introduced in its full glory. Of course, there are alternatives to racing and they are discussed here as well. Finally, aspects of racing that are in need of further exploration are pointed out.

Motivation

The development of approximate algorithms like iterated local search entails a selection among a potentially very large number of candidates. In Chapter 4 we will see that it is possible to define up to fourteen variants of iterative improvement and several hundred variants of variable neigborhood descent on the basis of three types of neighborhood moves and two types of pivoting rules alone. Each of these local search variants constitute a valid component of iterated local search

[1] cf. [Crow 79, Gold 85, John 02, More 02, Gent 97, McGe 96, McGe 99]

and so there are as least as many variants of iterated local search as there are variants of local search. In fact, without preselection, the actual number of variants to be considered, is likely to be several factors larger. For, in addition to local search, there are various ways to specify the acceptance criterion, the perbturbation mechanism, and the construction mechanism for the initial solution. All this wouldn't be a problem, if there would be clear–cut well–understood and thoroughly documented rules to guide the selection and composition of components. Alas, such rules do not exist.

In most applications of approximate algorithms, it is up to the developer to manually sift through the myriad of possibilities and some guidelines exist as to how to do this effectively, cf. [Best 01b]. However, in the context of this dissertation such an approach would work. To start with, the manual approach would be too tedious and time–consuming because the same process would have to be repeated for each of the twelve distinct problems introduced in Section 1.3 separately due to the all to real concern that the optimal algorithm composition be dependent on details in the problem specification [Culb 96]. In contrast, if the process of algorithm selection and configuration could be automated, then it would be possible to let the computer do the dirty work whilst the developer would have freed his hands to analyse rather than second guess the whealth of data generated by the computer and due to the standarised nature of the process, it would be easier to make comparisons at various stages in the development process. Finally, the availability of a standard automated tuning procedure could make it easier to compare experimental algorithms since developers would depend less on their fine–tuning prowess. Thus the fear that in comparing algorithms, the designer's aptitude rather than aptness of design is measured [Hook 96], could be alleviated.

Origins & Applications

Racing is a method that finds a good configuration from a given finite pool of alternatives by evaluating the performance of the members of the pool on a possibly infinite but in practice limited number of test cases. All candidates in the pool are tested on the first test case — and on the second, and as many as the experimenter deems necessary. Then, the candidates' performance is evaluated and those candidates that consistently under-perform relative to the pool as a whole are discarded. The remaining candidates are tested on additional test cases and, again, as soon as sufficient evidence has been gathered against specific candidates, these candidates are discarded. The racing continues until either only one candidate is left in the pool or as long as there exist additional test cases on which the candidates can be tested and the user-specified maximum number of experiments has not been reached.

Racing was first developed as a procedure to select models in memory-based supervised learning [Maro 94, Moor 94]. More recently, Birattari and others introduced racing as a method to select algorithmic configurations for combinatorial optimization problems [Bira 02, Bira 03, Bira 04]. Since then, racing has been applied with great success to the development of metaheuristics for combinato-

rial optimization problems in the context of the metaheuristics network [Chia 03b, Chia 03a].

Alternatives

Racing is not the only way to calibrate algorithms. Alternatives are calibration by analogy, calibration through trial and error, and calibration through search.

Calibration by Analogy Often, one can take advantage of past experience with the specification of algorithms [Laar 87, DeJo 90]. For instance, in case of iterated local search, it is known that the quality of the initial solution and the strength of the local search component should be high, while the perturbation should be sufficient yet not too strong and the acceptance criterion should not be too permissive. In case of algorithmic frameworks like ant colony optimization and genetic algorithms, one can go even further and let the algorithm design and calibration be guided by insights from (theoretical) biology.

Calibration by analogy works well if the goal is to obtain a decent configuration with little effort. However, there are two difficulties with this approach. The first is that one has to trust that past experience has been documented adequately. The second is that one has to trust that this past experience is transferable to the new problem domain to which the algorithm is applied. Moreover, the best that past experience can provide is bounds on parameter settings and a pre-selection of components. Additional testing is still needed to find the best configuration within this space [Culb 96]. For that purpose, racing is helpful.

Calibration through Trial & Error While some people rely on past experience to guide the development of algorithms, others rely on serendipity. That is, the development of the algorithm is allowed to be an haphazard process that depends on a constant flow of design ideas that the developer tests on a small number of test cases. The advantage of this approach is that it gives the developer a lot of freedom to invest all his or her creativity in the development process. The disadvantage of this approach is that it requires a lot of time and effort from the developer. Furthermore, the hit–or–miss character of the process makes that one can never be sure that one has found the best configuration out of all potential configurations.

A number of hands–off procedures exist for calibrating algorithms through trial and error. Genetic programming [Koza 92] is among the more famous. Note, however, that such hands–off procedures are bound to have disadvantages not dissimilar from the disadvantages of hands–on calibration. The procedures do not guarantee that they are able to find or generate the best configuration and rather than spending his or her time, effort, and experience on calibrating the algorithm, the developer is now required to spend this time and effort on calibrating the procedure for algorithm calibration. At the same time, it is harder to translate ideas in operators for the procedure than it is to immediately apply these ideas to the algorithm at hand.

Calibration through Search The third approach to calibration is to try to perform an exhaustive search and test each candidate a sufficient number of times on a sufficiently large number of training instances. This *brute force* approach has some drawbacks [Bira 02]: Firstly, the size of the training set is fixed a priori. Yet, there is no clear guide as to what constitutes a good size. On the one hand, employing too few instances makes it impossible to obtain reliable estimates of the candidates' performance. On the other hand, employing too many instances leads to a great deal of useless computation. Secondly, there is no clear guide as to how often tests in individual instances should be repeated in order to cope with the stochastic nature of metaheuristics. Finally, the same computational resources are allocated to each configuration so that manifestly poor configurations are as thoroughly tested as the best ones.

In comparison to brute force, racing algorithms provide a better allocation of computational resources to candidate configurations. By virtue of elimination of inferior candidates, racing is able to evaluate promising configurations on more instances and to obtain more reliable estimates of their behavior than brute force. Therefore, racing is more desirable in practice. Racing is not unique in this respect. For instance, [Coy 00] proposed a procedure, based on statistical design of experiments and gradient design, that finds effective settings for parameters in heuristics. [Aden 01] used Taguchi's partial factorial experimental design coupled with a local search procedure to find the fest values for up to five search parameters associated with a procedure under study. [Robe 98] used a fractional factorial experiment to set parameter values in neural network models for a finance application. [Park 98] used a nonlinear response surface optimization method based on a simplex design to find parameter settings in several applications of simulated annealing. [Pars 97] used statistical design of experiments to set the values of four parameters in a genetic algorithm. And [Xu 98], in the context of the Steiner Tree-Star problem, developed a procedure for fine-tuning five key factors in a tabu search heuristic. But racing is certainly one of the most elegant methods proposed so far.

Open Issues

Racing is a procedure that merits further research. Not only because the procedure has been applied with great success so far, but also because quite a few fundamental issues remain unsolved.

Theoretical Characterization The interpretation of the procedure's results is the first issue to be resolved. Racing is based on the sequential application of statistical tests. Whilst the behavior of the individual tests is well-understood, it is far less clear how to interpret the overall outcome of the application of these tests in sequence. That is, it is hard to indicate with what likelihood the racing procedure will discard candidates that should have remained in the pool and it is equally hard to indicate with what likelihood the racing procedure spends unnecessary CPU-cycles on candidates that should have been discarded earlier. That is,

how certain can be we be that the race does not produce false negatives or false positives? We can take comfort in the knowledge that similar problems are encountered in other branches of science. See for instance the discussion of model specification methodologies in [Kenn 00] and also [Love 83]. Yet, the fact that the problems remain unsolved casts a dim light on prospects in the case of racing.

Empirical Validation A Monte Carlo evaluation [Good 01] of racing is described in [Bira 02]. For this evaluation, 256 configurations of Max-Min-Ant-System were executed on 400 traveling salesman problem instances for 10 seconds on a 1.4GHz CPU with 512 MB of RAM. The solutions were evaluated and the evaluation values were stored in a 400×256 array. From the 400 instances, 1000 pseudo-samples were extracted. On each of the pseudo-samples a run of the racing procedure was simulated as follows: One after the other the instances are fed to the racing procedure. The procedure performs pseudo-experiments by reading the value for the instances–candidate pairs from the array and discards candidates on the basis of these values. Racing stops after executing 5×256 pseudo-experiments. The best candidate to come out of the race is then tested on 10 instances that were not used during the selection process itself. Thus, 1000 pseudo-samples yield a vector of 10×1000 components. Now, with this vector in hand, it was not possible to find significant differences between the quality of the candidates selected through racing and the quality of the candidates that would have been selected if all 400 instances would have been considered.

This is a nice result as it is. Nevertheless, it would be even nicer if similar results existed for similar experiments carried out with different algorithms on different problems so that we could get a feeling as to what extent the first results are generalizable.

Other Avenues In keeping with the spirit of metaheuristics, there is a whealth of options to be explored if one abandons the pretence of statistical validity. For instance, one could try out elimination schemes inspired by genetic algorithms. But also within the realm of statistics, there is room for experimentation. For instance, there are several methods to adjust for sequential testing. The question which method is best suited, is still open.

2.2 Performance Assessment

There are several ways to measure the performance of the candidates that race against each other in the racing procedure. Which one you employ depends on what you want to select. Consider the following analogy:

> It is widely known that computer programmers' pizzazz — their dynamism, their oomph, their zing — depends on pizzas and coke. With this dependence in mind, it is hard to fathom why computer scientists have devoted so little

energy to the study of either pizza or coke as there are few areas with more potential for a direct impact on programmers' performance than the quest for a better pizza recipe.

The pizza Margherita is the pinnacle of simplicity.[2] Next to pizza dough it has only three ingredients: Mozzarella cheese, tomatoes, and basil. The pizza owes its names to queen Margherita of Italy. At the time, Margherita was on a royal visit in the region around Naples. She had expressed her desire to become acquainted with the local food and so the commoners struck upon the idea to bake a pizza in the colors of the Italian flag — that is, a pizza with a grounding of red tomatoes, patches of white mozzarella and green basil on top. The queen was flattered and a famous pizza was born.

With the royal stamp of approval and in the footsteps of the Neapolitan exodus, the pizza Margherita swiftly conquered the world. Yet, the Margherita eaten by computer programmers over the world is but a far cry from the one that was originally approved. In an effort to stem the tide, recently a group of Neapolitan artisans has come together and petitioned the European Commission to become guardian of the Pizza Margherita Tradizionale (PMT). In order to qualify as PMT, the pizza dough should be made with fresh yeast, wheat, and sea salt; the tomatoes should be of the sweet San Marzano variety; the mozzarella should be made with buffalo milk; the basil should be freshly picked; and the pizza should be baked in a wood oven with a temperature of over $450°$ Celsius. Clearly, no one but the artisans who petitioned the commission can meet these requirements. All the more reason for consumers like us to try and find better alternative ways to experience the Margherita.

In order to find out which Margherita recipe suits us best, we need to vary the quantity and quality of the Margherita ingredients, bake the pizza and determine which variety has the best taste. Better still, we can rank the varieties according to their quality. Yet, although quality might be necessary, it is certainly not sufficient. For most practical purposes, we should also take the price into account. For instance, it is well known that the best Margherita is made with mozzarella di bufala from the town of Battipaglia, but if you happen to live in Tromsø, you may prefer to use the Norwegian mozzarella imitation as a substitute. More in general, rather than just looking at quality, one should try to pick the best pizza for any given price. On special occasions, like the rare date with his girlfriend, the programmer may be willing to spend a lot on an authentic PMT. On more regular occasions, like during a game of counter strike, a Margherita with Dutch green house tomatoes and Danish mozzarella look-a-like out of an electric oven may be just fine. The task of the assessor is therefore to pick those varieties of Margherita for which there do not exist other varieties that have an equal or higher quality for an equal or lower price.

There is one more complicating factor that the assessors should take into consideration: Pizzas, Margheritas included, can be cut and sold in parts. Assuming that the utility of Margheritas to programmers decreases with quantity — at some point they are not hungry anymore — divisibility implies that cheap

[2] I am indebted to Mauro Birattari for sharing his expert knowledge on this subject. All mistakes are mine.

> and greasy (c&g) pizzas that survive the competition with exquisite and expensive (e&e) pizzas sold per unit, may loose out when it becomes possible to spend a fraction of the original amount on a part. A programmer on a low budget cannot afford to buy a complete e&e pizza, but still prefers part of e&e to complete c&g. Hence, the assessor should not only take the cost–utility trade–off of a complete pizza into account, but also all other realizations of cost and utility that the pizza is capable of.

Thus, for pizzas quality is important, but price matters too. And if there are several combinations of price and quality for which the pizza can be sold, that should be taken into account as well. Likewise, for algorithms solution quality is an important criterion, but run time matters too. And if the algorithm can yield different solutions depending upon the amount of time it is alloted, that should be taken into account as well.

Solution Quality

The most straightforward way to measure the performance of an algorithm is to consider the solution that is returned by the algorithm and compute its score on the objective function. A slight complication arises when the algorithm is stochastic and different runs of the algorithm on the same test case might produce different solutions. When observing a stochastic quantity like solution quality, we are typically interested in its *expected value*. A single run of the algorithm on the instance at hand produces by itself an unbiased estimate of the expected value. If we were interested in a better estimate for that specific instance, we would consider the average of more runs in order to have a reduced variance.

With respect to the evaluation of the performance of an algorithm over a class of instances, it is useful to define performance as the expected value of the performance over the instances of the class, or, more roughly speaking, the average performance obtained on each instance, weighted by the probability that each instance occurs.

For our purposes, we are interested in measuring the performance of a range of algorithms so that we can select the best one. The generic selection procedure that we employ, racing, was described in Section 2.1. Imagine we wanted to find the best pizza Margherita, rather than the best scheduling algorithm. In that case, applying the racing procedure would be similar to baking a load of Margheritas according to different recipes and inviting a number of test-tasters to judge the pizzas. Every new test-taster constitutes a new test case for the pizza recipe and the recipes can be safely ignored for further tasting when it is disliked by the tasters that have tasted the pizza so far. For this perspective, the selection of the best pizza is a pure decision making process and so we can employ techniques developed in the field of decision analysis. These techniques are readily available [Ray 99]. Nevertheless, we will not use them. The reason for this is that we want to estimate to what extent the sample of initial test-tasters is representative for the whole population of Margherita consumers. That is, we want to be able to attach a level of certainty to the initial judgment. And so statistical tests are needed. In

our case, we follow [Bira 02], this statistical test is the Friedman two-way analysis of variance by ranks — Friedman test, in short.

For giving a description of the test, let us assume that the racing procedure has reached step k and that n candidates are still in the race. The *Friedman test* [Cono 99] assumes that the observed costs are realizations of k mutually independent n-variate random variables called *blocks* [Dean 99]. In this context, a block corresponds to the computational results of the candidates on a test case. Within each block, the costs are ranked from the smallest to the largest. Average ranks are used in case of ties. For each candidate j, let R_{lj} be the rank of j within block l, and $R_j = \sum_{l=1}^{k} R_{lj}$ be the sum of the ranks over all test cases. The Friedman test considers the following statistic:

$$T = \frac{(n-1)\sum_{j=1}^{n}\left(R_j - \frac{k(n+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}} \tag{2.1}$$

Under the null hypothesis that all possible rankings of the candidates within each block are equally likely, T is approximately χ^2 distributed with $n-1$ degrees of freedom. If the observed T exceeds the $1-\alpha$ quantile of such a distribution, the null is rejected, at the approximate level α, in favor of the hypothesis that at least one candidate tends to yield a better performance than at least one another.

If the null is rejected, we are justified in performing pairwise comparisons between individual candidates. Candidates j and h are considered different if

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k\left(1-\frac{T}{k(n-1)}\right)\left(\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}\right)}{(k-1)(n-1)}}} > t_{1-\alpha/2} \tag{2.2}$$

where $t_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the Student t distribution.

The Friedman test has a number of distinct advantages over other statistical tests. First of all, in contrast to for example the t–test, the Friedman test does not make any prior assumptions about the distribution of the observations. Furthermore, by focusing only on the ranking of the different candidates within each test case, the blocking as described above effectively eliminates the risk that the variation due to the difference among test cases washes out the variation due to the differences among candidates.

Solution Quality and Run Time

Often, we are not just interested in the quality of the solutions generated by the algorithm, but also in the speed with which these solutions are generated. If there are two algorithms and one of them yields good solutions in a matter of seconds while they other yields slightly better solutions in a matter of hours, then, for most practical purposes, the first algorithm is preferred over the second one. More in general, we wouldn't want to discard an algorithm as long as there is no other algorithm that performs better in terms of solution quality as well as in terms of run time.

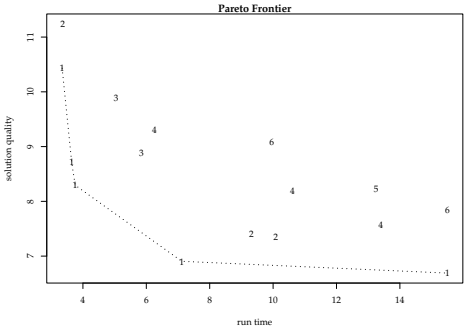


Figure 2.1: Bi-criteria selection.

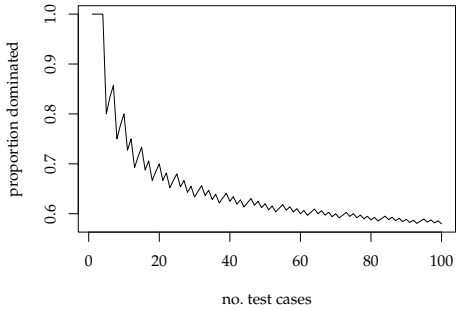


Figure 2.2: Binomial test.

Figure 2.1 illustrates a situation where we make a selection among 16 candidates on the basis of their performance on one test case. The performance is measured according to two criteria, run time and solution quality, and the position of the candidates on the grid indicates how well they did on each of the criteria. Now, the numbers on the grid indicate how many other candidates dominate the candidate represented on that position. In this context, a candidate on position (i, j) is said to dominate a candidate on position (k, l) if $i \leq k$ and $j \leq l$. The solid line on the plot connects those candidates that are only dominated by themselves. It is those candidates that we want to select.

Figure 2.1 shows that it is relatively straightforward to find the set of non–dominated candidates on the basis of a single test case: Just check them against all other candidates. When there are multiple test cases to base the selection on, things are more complicated however. Assuming there is some variety in the relative performance of the candidates, it will be likely that each candidate is going to be dominated by one of the others in one of the test cases. And an empty set is not what we want to select. Of course, we could take the Friedman test described earlier and apply it to both criteria separately. In that case we end up selecting only two candidates: The one that scored really well in terms of solution quality and the one that scored really well in terms of run time. However, candidates that struck a more moderate compromise between both criteria would be left out — and that is not what we want.

Another way to evaluate the candidates is to rank them according to the number of candidates that are dominating them. The problem with this approach is that it is biased towards candidates that operate in niches. When a candidate has many close colleagues, this candidate will sometimes manage to dominate these colleagues, but more often one of these colleagues will manage to dominate the candidate. On the other hand, a candidate that obtains a score on both criteria for which there is no close match by other candidates, will always be dominant. Hence the lone wolf will obtain an better rank overall than the wolf that stays in the pack. Consider we have to judge from a series of slight variations of Figure 2.1 rather than just this one picture. The candidate in the lower left corner would

probably get an overall rank of 1. Yet, the candidates in the upper left corner are likely to end up with a smaller overall rank. And so, when we rank according to the number of candidates that are dominating, only the candidate in the lower left corner would be selected. That is not necessarily what we want either.

The approach that I have eventually opted for, is to tabulate for each candidate how often it is dominated by another candidate. Candidates that are highly likely to be dominated by another candidate can be discarded. This is akin to a restaurant owner who asks each of his customers to pick the best pizza, but only compares tastes of customers with similar social backgrounds. The statistical test that is used to determine which candidates are dominated most of the time is the binomial test. The working of this test is illustrated in Figure 2.2 on the page before. As the number of cases on which the candidates are tested increases, the threshold for discarding candidates decreases. Figure 2.2 shows the development of this threshold. For the first five test cases, only candidates for which there exists another candidate that dominates them all the time are discarded. After ten test cases it is deemed to be sufficient if another candidate dominates the candidate 75% of the time and after 100 test cases this proportion has dropped to 60%. The threshold values correspond to the proportion of successes relative to the number of trials that is needed for the binomial test to assert with 95% confidence that the true probability of success is greater than 50%. That is, candidates are discarded, when one can be 95% confident that there exists another candidate that dominates them more than half of the time. In this way, the stochastically non-dominated set is selected. And this is exactly what we want.

Trace of Quality over Time

The third method to assess the performance of algorithm is to look at a set of possible outcomes rather than just one outcome. I will first establish why this makes sense and then discribe how such an assessment can be done.

Algorithm Traces A pizza has usually more than one pair of cost and quality associated to it. Rather than the whole pizza, you can buy $1/8^{th}$ part of the pizza for $1/8^{th}$ of the price — or a quarter or half or more than one pizza. The same thing is true for algorithms like iterated local search. Some end-users may be content with waiting a full hour to obtain a solution to their problem, other may want to get a solution within 10 seconds and are willing to accept that this solution may be of lower quality than the one found after one hour. When we assess the performance of iterated local search, this fact has to be taken into account. Since we do not know what kind of CPU-time the end-user may be willing to allocate, or since we are not willing to commit ourselves to a specific CPU-time, the whole range of cost-quality realizations has to be taken into account. In the context of iterated local search, a *trace* is the ordered sequence of run time and solution quality pairs of the solutions found by the algorithm. That is, each time a better solution is found, the algorithm's run time and the solution's quality are stored thus forming a trace of the algorithm's run.

Epsilon Indicator One trace is better than another one if, on the whole, its points are closer to the origin of the graph. There are several ways to assess the relative performance of traces and each have their particular weaknesses and strengths [Zitz 02]. In this research, I use a measure inspired by the ϵ–indicator. The ϵ–*indicator* [Laum 02] gives the factor by which a trace is worse than another with respect to both criteria, or, to be more precise: $I_\epsilon(A, B)$ equals the minimum factor ϵ such that for any solution in B there is at least one solution in A that is not worse by a factor of ϵ in all objectives. In practice, the ϵ value can be calculated as

$$I_\epsilon(A, B) = \max_{z^2 \in B} \min_{z^1 \in A} \max_{1 \leq i \leq n} \frac{z_i^1}{z_i^2} \tag{2.3}$$

In the single objective case, $I_\epsilon(A, B)$ is simply the ratio between the two objective values represented by A and B. The ϵ–indicator represents a natural extension to the evaluation of approximation schemes in theoretical computer science and gives the factor by which the outcome of an algorithm is worse than another [Erle 01]. In addition to that, it is cheap to compute.

Figure 2.3 on the following page provides the details for the ϵ–like measurement used to assess the relative performance of algorithms. The left panel of the figure consists of pseudo-code, the right panel contains plots to clarify what happens in the code.

Consider the plots first. The upmost plot depicts two traces, one made up of lower case letters, the other made up of upper case letters. The ϵ–indicator tells us by what factor the lower case letters have to be multiplied so that they no longer dominate any upper case letter.

Although many lower case letters dominate more than one upper case letter — for instance, b dominates A and B and f dominates C to G — the multiplication factor only has to be computed with respect to one upper case letter for each lower case letter. How it is determined which upper case letter matches to the lower case letter is shown in the middle plot. In the plot, the space below the trace of upper case letters is split into parts delineated by dotted lines. Any point within a part will no longer dominate any other upper case letter after it is multiplied by a factor computed with respect to the upper case latter at the upper right corner of the part. To see why this is the case, note that the dotted lines that delineate the parts also indicate the range of the direction in which the points will move after multiplication.

The multiplication factor does not even have to be computed for each lower case letter. The lowermost plot shows how the it is possible to determine the ϵ–indicator in two rather than in nine steps. The trick is to randomly pick one of the lower case points that dominates upper case points and compute the required multiplication factor for that point. Then, all lower case are multiplied by that factor and the procedure is repeated with respect to the points' new positions. After a few iterations no lower case point will dominate any upper case point anymore and than the value of the ϵ–indicator is found. In the lowermost plot, a

$\epsilon \leftarrow \mathsf{map}(\mathcal{C}, \mathcal{R})$:

1: $\epsilon \leftarrow 1$
2: **for** each i in $\mathcal{C}$ **do**
3: $\quad j \leftarrow \mathsf{match}(\epsilon \cdot \begin{bmatrix} i_x \\ i_y \end{bmatrix}, \mathcal{R})$
4: $\quad$ **if** $j \neq \emptyset$ **then**
5: $\quad\quad \epsilon \leftarrow \min\{j_x/i_x, j_y/i_y\}$

$q \leftarrow \mathsf{match}(p, \mathcal{S})$:

$\quad$ **if** $|\mathcal{S}| = 1$ **then**
$\quad\quad \{q\} \leftarrow \mathcal{S}$
$\quad\quad$ **if** $p_y \geq q_y \vee p_x \geq q_x$ **then**
$\quad\quad\quad q \leftarrow \emptyset$
5: $\quad$ **else**
$\quad\quad h \leftarrow \mathsf{head}(\mathcal{S})$
$\quad\quad t \leftarrow \mathsf{tail}(\mathcal{S})$
$\quad\quad$ **if** $p_y < h_y \wedge p_x < t_x$ **then**
$\quad\quad\quad$ **if** $p_x/p_y > h_x/\left(\mathsf{tail}(\mathcal{S} - \{h\})\right)_y$
$\quad\quad\quad$ **then**
10: $\quad\quad\quad\quad$ **if** $p_x/p_y < \left(\mathsf{head}(\mathcal{S} - \{t\})\right)_x / t_y$
$\quad\quad\quad\quad$ **then**
$\quad\quad\quad\quad\quad q \leftarrow \mathsf{match}(p, \mathcal{S} - \{h, t\})$
$\quad\quad\quad\quad$ **else**
$\quad\quad\quad\quad\quad$ **if** $p_y < t_y$ **then**
$\quad\quad\quad\quad\quad\quad q \leftarrow t$
15: $\quad\quad\quad\quad\quad$ **else**
$\quad\quad\quad\quad\quad\quad q \leftarrow \emptyset$
$\quad\quad\quad$ **else if** $p_x < h_x$ **then**
$\quad\quad\quad\quad q \leftarrow h$
$\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad q \leftarrow \emptyset$
$\quad\quad$ **else**
$\quad\quad\quad q \leftarrow \emptyset$

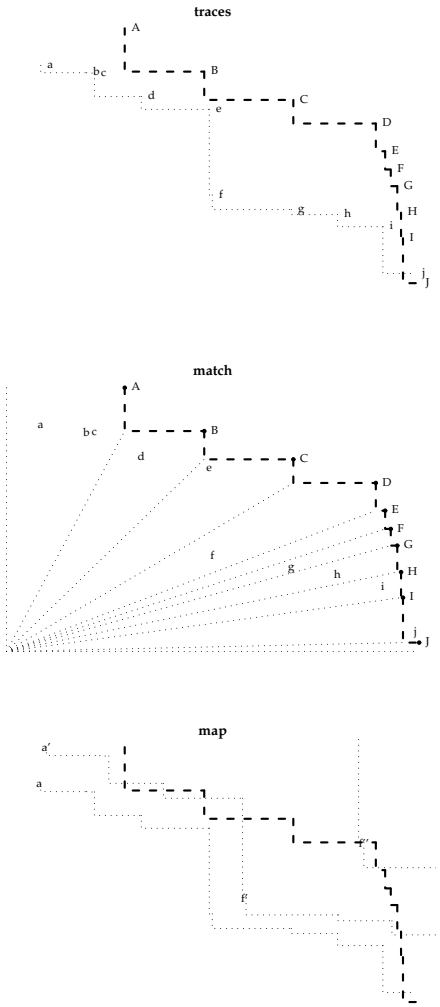


Figure 2.3: Computation of the ϵ–indicator: Implemenation & illustration.

was considered first and moved to a' so that it no longer dominated A. After that, b', c' and f', g' and h' still dominated an upper case point and so the procedure that to be repeated. This time f' was picked and moved to f''. After this multiplication, no lower case point dominated any upper case point any more and so it was found that $\epsilon = 1.74$ in for this pair of traces.

The computation of the ϵ–indicator is done with the functions map and match. The main function, $\mathsf{map}(\mathcal{C}, \mathcal{R})$, returns the value with which the pairs in trace $\mathcal{C}$ should be multiplied such that no pair in $\mathcal{C}$ dominates any point in trace $\mathcal{R}$ anymore. In order to find this value, map iterates through all pairs in $\mathcal{C}$, searches for a matching pair in $\mathcal{R}$ and computes the minimum of the ratios of the values in the pairs. The function $\mathsf{match}(p, \mathcal{S})$ that is invoked by map, returns the pairs of values on trace $\mathcal{S}$ that are dominated by p or it returns $\emptyset$ if no such pair exists.

Thus, ϵ is determined by repeatedly expanding one of the traces along the x-y-diagonal. When the expanding trace no longer dominates the other trace at any point, ϵ has been found. If both candidates' traces are equally good, or bad, then the value of ϵ is close to 1. If the expanding trace is much worse than the reference trace, then $\epsilon \gg 1$.

Reduction to Ranks The measure just discussed is fine if you want to compare the traces of two algorithms. However, in general, we would like to compare multiple algorithms. For that purpose, we can reduce the measurement outcomes to ranks as follows: For each pair of algorithms A and B, compute ϵ_1 for $\mathsf{map}(A, B)$ as well as ϵ_2 for $\mathsf{map}(B, A)$. Store the ratio of these two values in a matrix — ϵ_1/ϵ_2 in column A and row B; ϵ_2/ϵ_2 in column B and row A — and compute the sum of the columns. The column with the lowest sum corresponds to the algorithm with the most dominant trace.

Note a few points. First, the transformation of a ratio-matrix into an ordinal scale is a common transformation in decision science and the weighted sum method is one of the more straightforward ways to do so [Fish 70, Tria 98]. Second, computing all ratios of all pairs of candidates is expensive as it requires a number of calls to map which is quadratic in the number of candidates. In the multi–criteria decision making literature, several ways are discussed to reduce the number of comparisons without loosing much accuracy [Hark 87, Tria 99]. Furthermore, since we are only interested in good algorithms it may be worth to try and spare the effort that goes to computing the exact rank of algorithms that perform badly. Third, the ratio of ϵ values has an interesting interpretation. A ratio close to one indicates that the amount of zooming needed such that all realizations of algorithm A are dominated by a realization of algorithm B is about as big as the zooming needed in the reverse case. So the behavior of algorithms A and B is similar. On the other hand, if the ratio is close to zero or very large, algorithms A and B are very dissimilar. Additional information about the relative performance of A and B could be obtained by looking at the product of the ϵ values. A small value of the product indicates that the traces have a similar shape; a large value is indica-

tive of a widely different shape. Still, to really find out how traces of algorithms compare, it is necessary to plot and inspect each of them individualy.

Application in this Thesis

The three modes of performance measurement that we have looked at each play a role in a stage of the development cycle of iterated local search. In Chapter 3, where we have to make a selection among construction heuristics, it is appropriate to take only the solution quality into consideration since the construction of solutions requires very little time for most if not all methods and there is very little variation in computational effort due to differences in parameter settings within the construction heuristics. However, in Chapter 4, where we have to make a selection among local search algorithms, it is more appropriate to consider both, run time and solution quality, since there are considerable differences in run time between local search configurations and the additional run time does not always result in a better solution quality. When it does, it remains unclear whether the final iterated local search algorithm would not be better off with a local search component that yields a worse solution quality in less time. Finally, in Chapter 5, where we have to make the final selection of iterated local search algorithms, it is appropriate to take into account the dynamic behavior of the algorithm. For one, we should not commit ourselves to one particular run time, as it is unclear which one should be preferred. Moreover, if there are two varieties of iterated local search that obtain the same solution after, say, one minute, we should prefer the one that reaches the solution first.

With respect to the application of racing in the last phase of the development of iterated local search it should be mentioned that the racing operates on transformed data. After all candidates have been tested on a test case, the best overall solution quality reached is determined and for all solution quality realizations of all candidates in this test case, the square root percentage deviation from the best quality is computed. The motivation for operating on percentage deviations rather than raw values is that in this way the relative weight that the ϵ–indicator attaches to run time and solution quality becomes independent from the variations in solution quality ranges of the test cases. Furthermore, the square root of the deviations is taken to reflect the fact that differences among solutions close to the best one found are more important than differences far away from the optimum. Consequently, algorithms that make huge strides in the beginning and get struck soon after are discarded in favor of algorithms that find few improvements in the beginning but get better in the end.

One last note is in place concerning the classes of instances in relation to racing. The instances are randomly generated according to the same specifications as the ones that apply to the benchmark instance on which the fully developed algorithm is eventually tested. In principle, the instance classes to which it is applied correspond to the benchmark sets in that separate races are run for each benchmark set. In addition, for the calibration of the apparent urgency construction heuristic, racing is applied to subclasses of instances, where the subclasses are split accord-

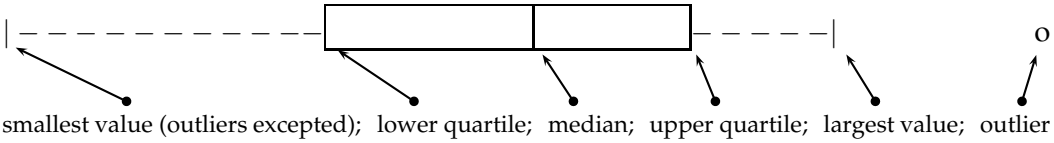


Figure 2.4: Example box–and–whisker plot with interpretation.

ing to characteristics of the due date distributions. Finally, in some cases racing is applied to instances for all classes taken together. This is to validate that the level of aggregation chosen has an effect on the outcome of the race.

2.3 Tools for Analysis

In Section 2.1, racing was introduced as an automatic hands–off procedure to select the best algorithms for the scheduling problems we are interested in. But it can do more than that. That is, the experimental results that were generated to serve the racing can be subjected to additional analysis so that we can get an idea as to why certain varieties perform better than others. This section catalogues a range of tools that we have at our disposal to aid the analysis, cf. [Cohe 95, Vena 99, Ripl 96, Main 01].

Visualization

Analysis usually starts by taking a look at the raw data. When there is a lot of data, visualization often helps to get an impression of general patterns.

The *box–and–whisker* plot, or boxplot, is useful if you want to compare groupings of data. Such a plot consists of a series of boxes like the one shown in Figure 2.4, one for each group. By putting these boxes–with–whiskers next to each other, on can easily see if groups are very different from each other. An instance where boxplots are particularly useful is when we want to compare the distributions of solution qualities generated by several algorithms such as in Figure 3.5 on page 57.

However, in many cases, we would like to compare groups in more than one dimension. For instance, because we are interested in how run time and solution quality interact. In these cases, *scatter-plots* are helpful (see e.g. Figure 4.2 on page 66). In some cases, the interaction of three variables may be what we are interested in. Then, a *level-plot* may do the job. However, when the third dimension is qualitative or discrete, then replacing hues of gray with symbols may actually give a better result (see e.g. Figure 3.6 on page 59).

Clustering

Given that this dissertation reports on the application of a wide variety of candidate algorithms to a wide variety of problem instances, it might be useful to try and map these varieties. That is, it would be nice if we could come up with a taxonomy of candidates or a taxonomy of problems purely based on the experimental

results.

Figure 2.5 on the next page is an example of the kind of picture we would like to obtain for candidate algorithm or scheduling problems. This particular tree represents a taxonomy of a selection of Darmstadt's pizzerias. Say this tree was generated on the basis of customer data for each restaurant (which is not the case), then from the tree we learn that Pizzeria da Nino and Pizzeria da Giuseppe draw from a similar customer base since they are located close to eachother on the tree. Similarly, from its location at the opposite end of the tree we can infer that Joey's Pizza Service services a completely different type of customer than Pizzeria da Giuseppe. Meanwhile, from the location of Pizzeria Rotkäppchen, we can infer that its customer base is more closely related to Pizzeria da Nino and Pizzeria da Giuseppe than to Pizza Taxi or Joey's Pizza Service, but at the same time, the customers of Pizzeria da Nino and Pizzeria da Giuseppe have more in common than the customers of Pizzeria Rotkäppchen and Pizzeria da Nino.

Clustering [Ever 01] allows us to generate trees like this. The method employed in this dissertation is hierarchical agglomeration. In *hierarchical agglomeration* each observation starts as a separate group. Groups that are "close" to one another are then successively merged. The output yields a hierarchical clustering tree, known as a *dendrogram*, that shows the relationships between observations and between the clusters into which they are successively merged. Hierarchical methods avoid specifying how many clusters are appropriate by providing the user with many different partitions by cutting the tree at some level. There are several ways to merge clusters. In this dissertation only complete–link clustering is considered. *Complete–link clustering* joins two clusters if and only if all members of one cluster are close to the other cluster, and so tends to produce 'compact' clusters, and relatively similar objects can remain separated up to quite high levels in the tree.

Some data normalization is required to make clustering work. In the case of algorithm clustering, the algorithm scores on the instances — that is, the objective function values of the solutions obtained — are scaled to make sure that each instance is weighed equally by the clustering procedure. The scaling is done by dividing the algorithm's score on the instances by their root mean square and subtracting the mean. After scaling, the scores are transformed into a distance matrix denoting the Euclidean distances between the scores of the algorithms. With this matrix in hand, hierarchical agglomeration merges the algorithms in clusters and the clusters in clusters of similar clusters. The dendrogram plots that are generated this way show the dissimilarity at which clusters are merged on the vertical scale. Hence they show the construction process from bottom to top (see e.g. Figure 4.8 on page 75).

Tree–based models

Clustering tells us which candidates or instances are similar, but it doesn't tell us why they are similar or in what respect they are similar. When we want to find out the answer to questions like that, tree induction [Brei 84] is what we need.

Just like hierarchical partitioning, tree induction yields a tree of the type shown

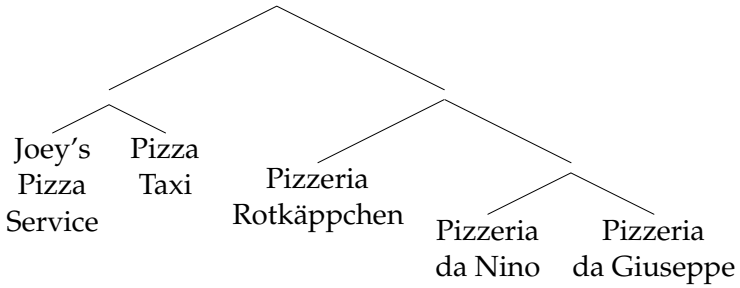


Figure 2.5: Taxonomy of pizza restaurants.

in Figure 2.5. The difference between the two methods is that with tree induction a label will be attached to each split that indicates on what basis a particular split is made. The label at the root will be the splitting criterion that has the best predictive value and the other labels specify the subsequent delineations. So, in case of Figure 2.5 the splitting criterion at the root could be home or away and the subsequent distinction between Joey's Pizza Service and Pizza Taxi the time that customers are prepared to wait. Rotkäppchen could distinguish herself from Da Nino and Da Giuseppe in the level of the pizza prices and Da Nino differs from Da Giuseppe in its proximity to the University. The point is that tree induction makes a selection out of a various splitting criteria and shows which yield the highest predictive value.

Tree induction or *recursive partitioning* as it is also known, is the process of building a tree based model such as a classification tree or a regression tree on the basis of a set of examples. An example of a regression trees is a decision tree that links the quality of the solution found by a candidate on a particular test case to characteristics of this candidate. In a classification tree, the path from root to leaf would link characteristics of the candidate to the class to which it belongs. Tree induction is a supervised process in that for each example the required answer is given in advance. So, for the construction of a classification tree of algorithmic candidates, we have to determine in advance to which class each candidate belongs, for instance through clustering, and list the characteristics of all candidates. On the basis of this information, a tree can then be constructed by successively splitting the candidates into groups on the basis of common characteristics. The nodes on the tree denote the splitting criteria and the leaves on the tree contain candidates with a common classification. In the regression tree example above, the leaves indicate the solution quality that one or more candidates have in common and the nodes indicate what candidate features induce candidates to yield this particular quality. Figure 4.10 on page 79 is an example of such a tree.

What size the tree is going to have is up to the user. In rpart [Ther 97], the method for tree induction that is used here, the user can specify how complex the tree should be by setting a parameter called *cp*. The value of *cp* is between zero and

one. The closer to zero *cp* is set, the larger the tree.[3] In general, the size of the tree does not matter very much. It should be large enough to contain some interesting information and small enough so that it can still be displayed on a page. However, tree based models can also be used for predictive purposes. For instance, in Section 3.1 a tree is described that predicts how the apparent urgency heuristic should be tuned depending on features of the particular instance it is supposed to solve. In this case, size does matter: A tree that is too large may reflect the data perfectly, but because of that it may also reflect aspects of the data that are peculiar to the particular sample that the construction is based upon and hence a smaller tree may actually be better at predicting outcomes for new examples. This phenomenon is known as *overfitting*. In order to obtain the right size, the tree has to be pruned. If a separate validation set is available, we can predict outcomes for that set and determine how much the predicted outcome deviates from the actual outcomes in the validation set for trees of different sizes. The best tree is the smallest tree with the smallest deviation. In rpart, one–tenth of the data are set aside as test set and cross validation errors are determined with respect to this set. On the basis of these data the user can then decide which value of *cp* he or she prefers.

Linear Models

A more conventional way to go about analyzing data is to construct a linear model of the candidates and the instances and to subsequently try and fit this model to the data [Fara 02]. The model would have the following form:

$$y = \alpha + \sum \beta_i x_i + \epsilon, \tag{2.4}$$

where y is the variable to be predicted, say the solution quality, x_i is the i^{th} variable to be used in the prediction, α and β_i are coefficients that need to be estimated, and ϵ is the error term. Several procedures can be employed to estimate the values of α and β_i. Ordinary least squares is what is normally used.

Since there is not a great deal of established theory with respect to the application of iterated local search to scheduling, we cannot be quite sure that the models are accurate. Luckily, there exist several methods that search through the space of potential models and return those models that score best according to some sort of criterion like R^2 or Aikake's information criterion [Aika 74]. [Mill 90, Hoet 99] discuss those methods.

Survival Analysis

Racing is more than just a series of experiments. It is also one big experiment itself. The application of racing to algorithms is akin to the application of a poison to patients. Some will survive, most will die. If you carefully log who dies when, you can apply a host of tools on the log afterwards. These tools are part of *survival analysis*. At its simplest, survival analysis consists of plotting the survival rates of candidates or groups of candidates over time.[4] In the context of racing, the flow of

[3] cf. [Ripl 96, §7.1]

[4] In the survival analysis, groups are usually called *strata*.

time corresponds to the addition of extra test cases, and if a candidate is no longer tested, it has died.

Apart from these basic surival plots, it is possible to construct trees to predict the survival rate or to fit a linear regression model for the same purpose [Ande 82].[5]

2.4 Summary

The selection of algorithms is done with the help of a racing procedure. Racing is an iterative process in which candidates are evaluated on test cases. Candidates that consistently perform badly are removed early on and in this way more resources can be allotted to the assessment of more promising candidates. Three useful ways to assess the performance of algorithms are (i) to look at the quality of the solutions they yield, (ii) to look at the trade–off between run time and solution quality they offer, and (iii) to look at all realizations of run time and solution quality they offer. All three modes of assessment are employed in various stages of the algorithm development described in Part II. As for analysis, a plethora of tools from statistics is employed. Among them are cluster analysis, survival analysis and recursive partitioning. The hope is that all this will lead to well justified algorithm configurations and will contribute to a better understanding of the reasons behind the algorithms' performance.

[5] There also exist methods to search through space of linear models in S [Voli 97] which are similar to the procedures use for ordinary linear regression models, but unfortunately I couldn't get them to work on R, the statistics package employed here.

Part II

Development

3 Construction

In this chapter, we examine ways to construct an initial solution on the basis of problem instance data. Obtaining an initial solution of high quality is important because more often than not the quality of the initial solution will determine how easy it is for programs that take this solution as starting point to obtain subsequent solutions of an even higher quality.[1] For the scheduling problems we are concerned with, solutions are represented by a sequence of jobs. Consequently, construction is equivalent to the assignment of an order to the jobs. There are basically two methods for construction. Construction method one assigns a score to each job on the basis of the job's due date, processing time, and weight and sorts the jobs according to their score. This method, known as dispatching, is examined in Section 3.1. Construction method two differs from construction method one in that method two evaluates partial solutions using the problem's objective function and inserts jobs at the position in the partial solution that yields the best objective function value.[2] This method of construction by insertion is examined in Section 3.2. Both examinations involve a great many computational experiments that are described in some detail and so for everyone's convenience, a summary of the main findings is given in Section 3.3.

3.1 Dispatching Rules

Dispatching is a fast way to come up with an initial solution as the job scores on the basis of which the jobs are ordered tend to be easy to compute. This section first surveys several scoring functions. Following that, it describes the implementation of one of the more intricate and successful scoring functions known as *apparent urgency*. Apparent urgency is an atypical dispatching rule in that it is parametric. This section investigates what parameter setting should be chosen in order to obtain optimal performance; it explores a variety of models that explain the parameter choice; and it investigates which of the models predicts the parameter settings that yield the best performance for apparent urgency.

[1] That is, a higher quality than the quality of the solutions that would have been obtained if the programs would have start with a worse initial solution.

[2] As we are dealing with minimization problems, that is the lowest objective function value in our case.

Concepts

There are several ways to assign a score to a job. One way that is often used is to assign a random value to the job. This is useful when nothing is known about the job or when a wide variety of initial solutions needs to be generated. However, the assignment of random values tends to yield solutions of low quality. Often, simply sorting jobs according to their due dates — or processing times, or weights, or any combination of these data — yields solutions of a significantly better quality. Dispatching rules based on scoring functions like these are known as *static* dispatching rules since the scores are determined completely by the instance data. In contrast, *dynamic* dispatching rules also take into account data that change as the solution is built. An example of this kind of dynamic data is the *makespan* of a partial solution. While the makespan is low, it is likely that a job that is added to the partial solution will complete before its due date. When the makespan becomes larger, the likelihood that the job completes in time shrinks. In fact, with knowledge about the makespan of a partial solution and the job's due date we can get exact information as to whether the job will be late or not. With this in mind, it is clear that the incorporation of dynamic data can be a useful element of dispatching rules. Next to static and dynamic dispatching rules, there is a third class of dispatching rules known as composite dispatching rules. *Composite* dispatching rules assign a score to jobs on the basis of a weighted sum or weighted product of the scores of these jobs in two or more elementary dispatching rules. An example of this third class is apparent urgency which is discussed next.

Implementation

For the single machine total weighted tardiness problem ($1||\sum w_jT_j$), apparent urgency is reported to be the best dispatching rule available [Pott 91, Koul 94, Koul 98].

Apparent urgency (AU) [Mort 84, Mort 93], which is also known as apparent tardiness cost [Pine 95], is a composite dispatching rule which orders jobs according to

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k\overline{p}}\right), \tag{3.1}$$

where p_j, d_j, and w_j are processing time, due date and weight of job j, t is the sum of processing times of the jobs that have been scheduled sofar, $\overline{p}$ is the average processing time of the remaining jobs, and k is a parameter.

AU is a combination of two elementary dispatching rules. One orders jobs according to weighted shortest processing time (WSPT) — that is, according to w_j/p_j. The second orders jobs according to minimum slack (MS) — that is, according to $\max(d_j - p_j - t, 0)$. MS is useful for a problem like $1||L_{\max}$. WSPT is useful for a problem like $Pm||\sum w_jC_j$ and WSPT is yields to optimal solution fo $1||\sum w_jC_j$. AU's only parameter, the look ahead parameter k, determines the relative importance of WSPT and MS. If k is very large, the AU rule reduces to the WSPT rule. If k is very small, the rule reduces to the MS rule for early jobs and the WSPT rule for late jobs.

AU was originally developed for the single machine environment. To make it work in parallel machine and flow shop environments some minor adjustments are needed in order to ensure that the effect of adding a job to a partial solution is evaluated appropriately.

Apparent urgency for the single machine environment is implemented according to the following equations:

$$\mathsf{AU}(t, \mathcal{S}) = \begin{cases} \mathcal{S} & \text{if } |\mathcal{S}| \leq 1 \\ \{j^*, \mathsf{AU}\,(t + p_{j^*}, \mathcal{S} - \{j^*\})\} & \text{otherwise} \end{cases} \tag{3.2}$$

$$j^* = \arg\max_{j \in \mathcal{S}} I_j(t) \tag{3.3}$$

Here, t is a variable that represents the makespan of the partial solution and $\mathcal{S}$ is the set of jobs to be scheduled. Equation 3.2 is a recursive definition of the procedure. Starting with $\mathsf{AU}(0, \mathcal{S})$, it will call itself until all jobs in $\mathcal{S}$ are scheduled and $t = \sum_{j \in \mathcal{S}} p_j$. Equation 3.3 defines j^*, the next job selected by AU, as the job with the maximum score on function $I_j(t)$ which was given in Equation 3.1.

For the parallel machine environment, the score function $I_j(t)$ remains the same. However, the general procedures have to be adapted as follows:

$$\mathsf{AU}(\mathcal{T}, \mathcal{S}) = \begin{cases} \mathcal{S} & \text{if } |\mathcal{S}| \leq 1 \\ \{j^*, \mathsf{AU}\,(\mathcal{T}', \mathcal{S} - \{j^*\})\} & \text{otherwise} \end{cases} \tag{3.4}$$

$$t' = \min \mathcal{T} \tag{3.5}$$

$$j^* = \arg\max_{j \in \mathcal{S}} I_j(t') \tag{3.6}$$

$$\mathcal{T}' = T \backslash \{t'\} \cup \{t' + p_{j^*}\} \tag{3.7}$$

Here, $\mathcal{T}$ is the set of m completion times of the last jobs scheduled on the m machines in the environment. At the start, all completion times are zero and each iteration, the smallest $t \in \mathcal{T}$ is incremented with the processing time of the last job that is added to the schedule (Equation 3.7).

Also in the flow shop environment, function $I_j(t)$ is valid. The wrapping procedure is like the AU procedure from Equation 3.4, albeit subject to the following amendments:

$$j^* = \arg\max_{j \in \mathcal{S}} I_j(\max \mathcal{T}) \tag{3.8}$$

$$\mathcal{T}' = \{\forall t_i \in \mathcal{T} : t_i \leftarrow \max(t_i, t_{i-1} + p_{i-1,j^*}) + p_{i,j^*}\} \tag{3.9}$$

Equation 3.8 states that the job score is computed relative to the completion time of the last job on the last machine in the current schedule. Equation 3.9 spells out the update of completion times after the addition of job j^*. Here, p_{i,j^*} is the processing time for job j^* on machine i.

Calibration

Apparent urgency performs best when an adequate choice is made regarding the value for the look-ahead parameter k that determines how the priority assignment

formula weighs the job's weighted processing time against its slack. Recall that if k is very large, the AU rule reduces to the WSPT rule and if k is very small, the rule reduces to the MS rule for early jobs and the WSPT rule for late jobs. Now, WSPT is a popular heuristic for $1||\sum w_j U_j$,[3] and so one can probably safely set k to a large number such as 5. A value of $k = 2$ has been used in static flow shops with the $\sum T_j$ objective [Veps 85]. For the $1||\sum w_j T_j$ problem, AU is reported to perform well if k is set to $k = 0.5$ for $TF = 0.2$, $k = 0.9$ for $TF = 0.4$ and $k = 2.0$ for $TF > 0.4$ [Pott 91, Cong 02] (TF is the tardiness factor, an indicator for the due date distribution defined in Section 1.3 on page 10). For the other problems, it is unclear what value k should have.

In order to find out what k values to choose, 600 racing experiments were carried out on 600 distinct classes of instances. The total of 600 classes was obtained by splitting the 24 instance classes define in Section 1.3 on page 15 into 25 classes each. The 25 subclasses distinguish themselves from each other in the distribution of the due dates. That is, each individual subclass corresponds to a pair of tardiness factor and range of due dates from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, where the tardiness factor and the range of due dates are the indicators of due date distribution that were defined in Section 1.3 on page 10. In each race, 41 distinct values for k are tested using $k \in \{2^\kappa : -10 < \kappa < 10, 2\kappa \in \mathbb{Z}\}$. The race continues until all but one k value have been discarded, or 410 evaluations have been carried out, or 100 instances have been used as test case. The first k values are discarded from further consideration after the values have been tested on at least 5 test cases. Algorithm performance is measured in terms of the solution quality only (as described in Section 2.2 on page 31).

Figure 3.1 on the next page summarizes the results of the 600 races. The figure is composed of 24 plots, one for each main class. In each plot, the 25 κ values selected by the 25 races are delineated by a solid line. The dashed line delineates the maximum κ values that were still not discarded at the end of the race and the dotted line delineates the minimum κ that were still in the race. The 25 κ values are ordered according to tardiness value (TF) and range of due dates (RDD) as follows: $(TF, RDD) \in \{(0.2, 0.2), (0.4, 0.2), \ldots, (0.8, 1.0), (1.0, 1.0)\}$. Vertical lines are added to each plot to make indicate where the value of *RDD* changes. That is, in the part between the first pair of vertical lines, $RDD = 0.2$ and TF is a sequence of 0.2, 0.4, 0.6, 0.8, and 1.0; in the second part, the TF sequence is repeated for $RDD = 0.4$; and the third, fourth and fith part correspond to $RDD = 0.6$, $RDD = 0.8$, and $RDD = 1.0$ repspectively. Thus, from the plot in the upper left corner of Figure 3.1 we can deduce that in case of $1||\sum T_j$ the races have been able to pick a clear winner since the lines for the minimum, best and maximum κ hardly diverge. Furthermore, in that plot, the value of κ tends to be smaller for extreme values of TF and gradually drops as *RDD* increases.

The patterns from the plot in the upper left corner re–emerge elsewhere in the

[3] However, worst-case analysis shows that WSPT may perform arbitrarily bad on this problem [Pine 95, §3.3].

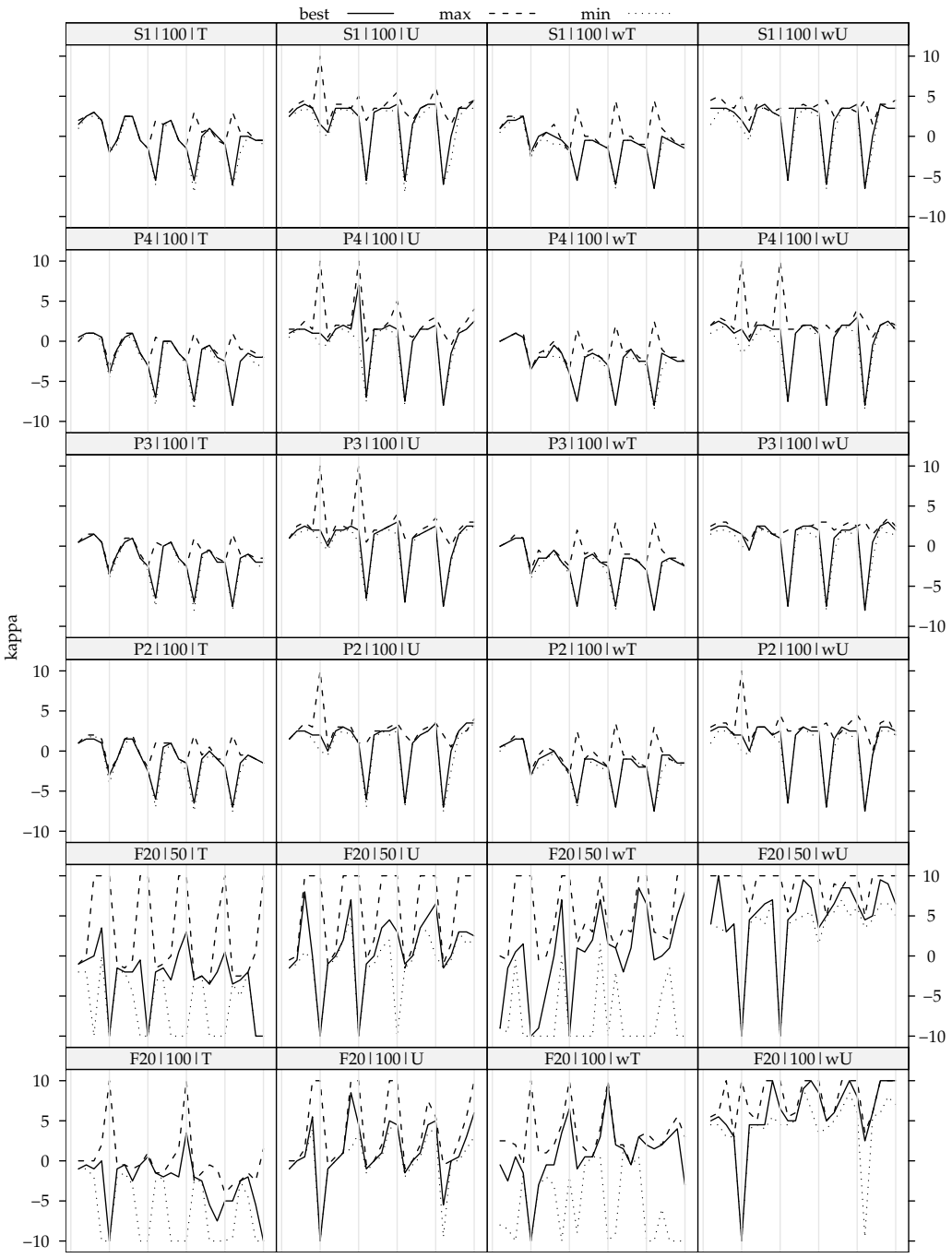


Figure 3.1: Best, minimum, and maximum κ values for apparent urgency selected in the races (kappa $= \log_2 k$).

figure. In the aggregate, Figure 3.1 suggests that the best value of k increases as TF approaches 0.5 and given that, apart from perhaps the flow shop problem instances, the minimum and maximum values for κ tend to be close to the best value for κ, it is likely that the selection of κ is pretty precise. Note also how the value of RDD seems to have an effect in the context of the total tardiness minimizing problems in columns 1 and 3 but not in the context of the total unit penalty minimizing problems in columns 2 and 4 as the trend of κ–values is downwards on the former columns whilst it is constant in the latter.

Modeling

Table A.1 and Table A.2 contain the κ values that resulted to be the best in the races on which Figure 3.1 is based. Given that these data are available, the logical next step is to test any theories we might have about the look-ahead parameter against these data. However, there is hardly any theory on this subject. We know that the best k value probably depends on the problem specification and the distribution of the due dates, but how exactly is unclear. Therefore, here the "kitchen sink" approach is more appropriate.[4] That is, we lump everything that might influence the optimal look-ahead value together and then progressively weed out anything superfluous. In the case at hand, problem environment (single, parallel or flow shop), objective (tardiness or unit penalty), weight (yes or no), number of jobs and number of machines might matter. In addition, tardiness factor (TF) and range of due dates (RDD) might be used as indicators of problem hardness. The final model will consist of a combination of a subset of these variables.

In addition to variables one has to choose a syntax with which to construct the model. Here, we consider two variants introduced in Section 2.3. The first variant is a syntax for linear models of the form $y = \alpha + \sum \beta_i x_i + \epsilon$ where y is the variable to be explained, α is a constant, the x_is are explanatory variable and the β_is are coefficients; ϵ is a rest term covering random noise, disturbance, or error. The second variant is a syntax for decision trees, where the nodes are choice variables and leafs are the value predictions. Linear models are interesting because they conform to the conventions of algebraic formula notation. Decision trees are interesting because they resemble more closely how one would implement a computer algorithm. Linear models are generated with the regsubsets procedure in R [Mill 90]. This procedure tests all combinations of variables and returns for any number of variables, the model that obtains the best fit to the data. Decision trees are generated with the rpart procedure in R [Brei 84]. This procedure returns one decision tree that can be pruned so that one can decide *a posteriori* how big the tree should be. For both variants, more variables or nodes entail a better fit. But the best fit is not necessarily the best choice since over-fitting may occur because of some bias in the instances on which the race was performed and because the races are not 100% guaranteed to yield the optimal outcome and so extra experiments might be needed in order to make the final selection.

Table 3.1 and Figure 3.2 list a number of linear models and decision tree models.

[4] See also the discussion on modeling in Section 2.3

Table 3.1: Regression subsets: The values in the fields are the coefficients for the variables in the regression subsets.

	α	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9	β_{10}	β_{11}	β_{12}	β_{13}	β_{14}
1	-1.6	0.6	6												
2	-2.2	0.6	6	-0.042	0.3										
3	-2.8	4.3	10	-0.042	0.3	-18	-23								
4	-1.5	4.3	10	0.004	0.3	-18	-23	-2							
5	1.4	2.0	8	0.004	0.3	-32	-42	-5	30	41					
6	1.1	2.5	8	0.066	0.4	-28	-38	-6	31	41	-0.7				
7	0.6	3.4	9	0.028	0.3	-22	-32	-5	16	26	-1.7	2			
8	1.9		6	0.028	0.3	-14	-23	-9	-3	7	-1.7	2	8		
9	1.1		6	0.028	0.3		-13	-6	25	40	-2.2	3	5	-42	
10	1.4		4	-0.058	0.3		-11	-6	26	39	-2.1	3	6	-43	0.2

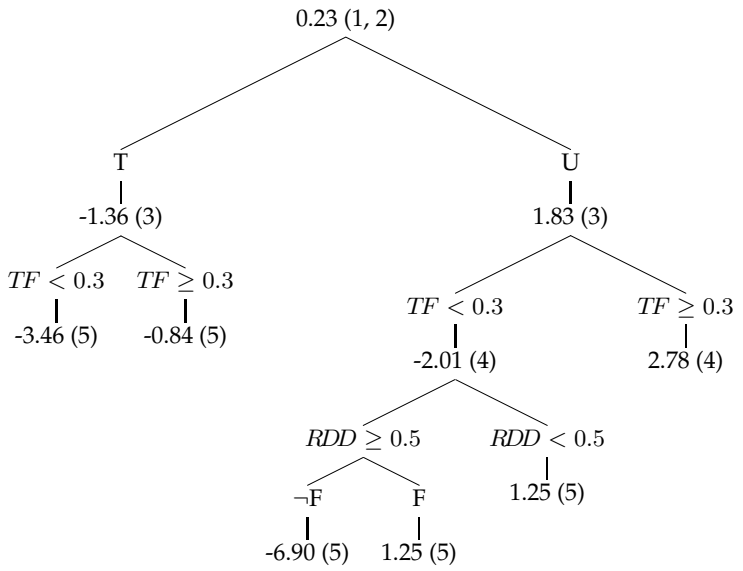


Figure 3.2: Decision tree used to determine the value of κ for apparent urgency. The number in parentheses is an indication of the complexity of the (sub) tree.

In Table 3.1 on the preceding page, the coefficients for 10 distinct linear models are listed. The variables employed in these models are drawn from array $X = [TF \cdot T, TF \cdot U, RDD \cdot \neg w \cdot m, RDD \cdot w \cdot m, (TF - 0.5)^2 \cdot T, (TF - 0.5)^2 \cdot U, RDD, RDD \cdot TF \cdot (TF - 0.5)^2 \cdot T, RDD \cdot TF \cdot (TF - 0.5)^2 \cdot U, TF \cdot (TF - 0.5)^2 \cdot m, RDD \cdot TF \cdot (TF - 0.5)^2 \cdot m, RDD \cdot TF, RDD \cdot (TF - 0.5)^2, RDD \cdot U \cdot m]$, where x_i is the i^{th} variable in X and β_i is the coefficient associated to that variable. For instance, linear model 1 is defined as follows:

$$\kappa = \alpha + \beta_1 TF \cdot T + \beta_2 TF \cdot U + \epsilon \tag{3.10}$$

and linear model 10 is defined thus:

$$\kappa = \alpha + \beta_2 TF \cdot U + \beta_3 RDD \cdot \neg w \cdot m + \beta_4 RDD \cdot w \cdot m + \ldots + \beta_{14} RDD \cdot U \cdot m + \epsilon \tag{3.11}$$

The models in the table are the ones that obtain the best fit to the data with 1, 2, 3, ..., or 10 variables. Variables like $TF \cdot T$ and $TF \cdot U$ are complementary in that both represent the interaction between tardiness factor (TF) and objective function (T or U). The variables in the models in Table 3.1 are selected from the problem specification variables U and T (objective), w and $\neg w$ (weight), m (number of machines, and, implicitly, machine environment), n (number of jobs), TF (tardiness factor), RDD (range of due dates), $(TF - 0.5)^2$ (a transformation of tardiness factor that suggested itself in preliminary examinations), and any interaction between these variables. What can be concluded from the table is that the due date distribution implied by TF and RDD in interaction with other elements of the problem specification determines the value of κ. The values of the coefficients in the table should be taken with a grain of salt. It is likely that the estimation of these values is biased since most models are likely to violate the assumptions of ordinary least squares regressions. in particular, one would expect that the data mining approach of regsubsets is prone to introduce misspecification errors which bias the estimates of the coefficients either because not all sources of variation are covered by the model or because the same source of variation is covered by more than one variable.

Figure 3.2 on the page before gives the root and the main branches of a decision tree generated on the basis of tardiness factor (TF), range of due dates (RDD), machine environment (F, P, S), objective (T, U), weight (w), number of machines (m), and number of jobs (n) of the problem instance. The number in between braces indicates at what stage the branches would be cut off when rpart's prune method is invoked. The number corresponds to a value of parameter cp that specifies what complexity the tree is allowed to have relative to its performance (see also Section 2.3). The number i in between braces corresponds to the value 2^{-i} for this parameter. The tree in Figure 3.2 is pruned with $cp = 2^{-5}$. Figures A.1 to A.3 on pages 122–123 give the extensions to this tree if the value is allowed to decrease from $cp = 2^{-6}$ to $cp = 2^{-10}$. The numbers on the leafs that are not between braces are the values of κ that should be chosen according to the decision tree. These κ values stand for k-values of 2^{κ} in the apparent urgency rule. Thus,

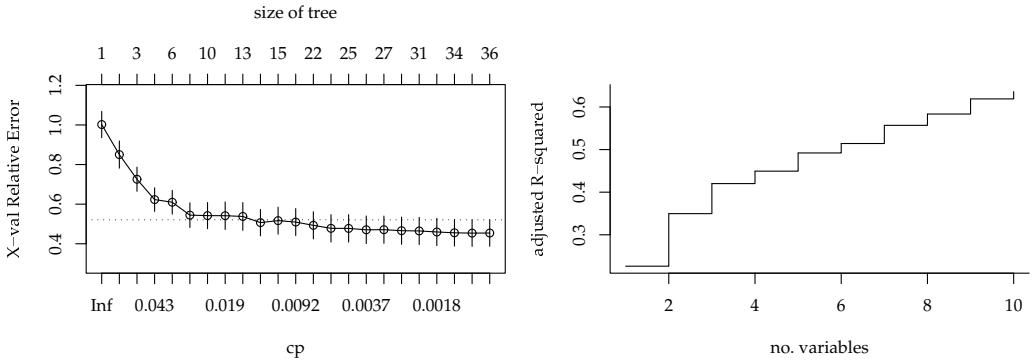


Figure 3.3: Goodness of fit: On the left, the cross-validation error is plotted against the tree–size; on the right, the adjusted R^2 is plotted against the number of variables in linear models.

the value of 0.23 at the root indicates that the default value of k should be 1.17. When the objective function is to minimize total tardiness, i.e. on the left branch from the root node, k should be $2^{-1.36} = 0.39$ else it should be 3.56. Furthermore, k should be small when the tardiness factor is small. Overall, like in the regression subsets, in decision trees the due date distribution tends to be the most important element in deciding what value k should take.

Selection

While it is nice to have all these models summarized in Table 3.1 and Figure 3.2, ultimately we would want to select one of them to determine the values that the k parameter should take in the application of the apparent urgency rule. Figure 3.3 provides a graphical means to select such a model. The figure shows in its left panel how the cross–validation error of the decision tree on the race data decreases as more nodes are included in the tree. In the right panel, it shows how the fit of the linear model improves as more variables are included. In both cases, more is better. However, it might be the case that attempting to obtain an optimal fit to the k values selected by the 600 races is counterproductive. After all, perhaps some races erred in selecting the k–values and then trying a model that could exactly predict these k–values could possibly lead to worse results than a slightly more sloppy model. In other words, a big tree runs more risk of overfitting than a small tree. Moreover, the cross-validation errors of decision trees are not readily compared against the adjusted R^2 scores of linear models.[5]

But, of course, we can use the racing procedure for model selection described in Section 2.1. Earlier in this section, it was described how racing is employed to determine the best k values for AU. Now that we have models of k, we can have each model represent a candidate and employ racing to select the best candidate

[5] Adjusted R^2 is the proportion of the variability in one series that can be explained by the variability of one or more other series in context of regression adjusted for the degrees of freedom in the model.

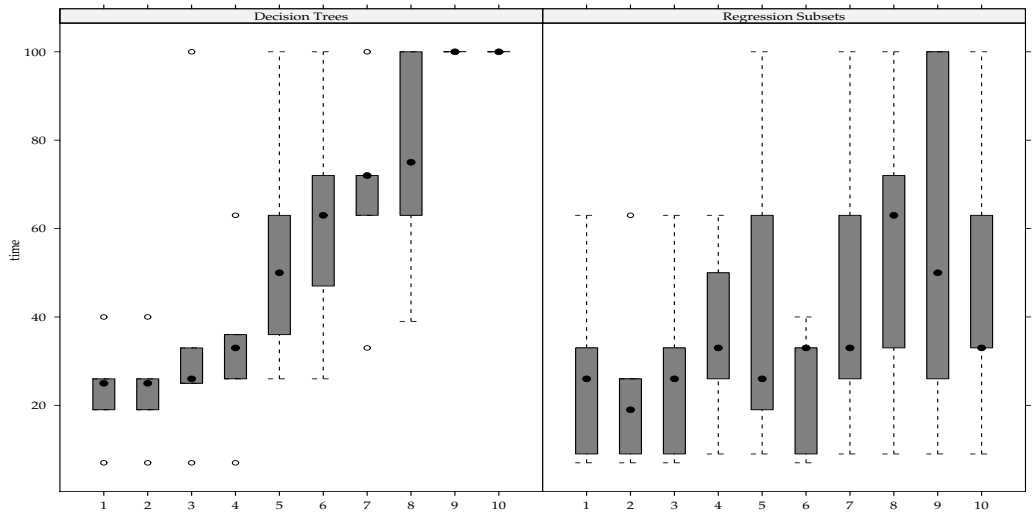


Figure 3.4: Racing survival of calibration models.

among them. The cases on which the models are tested is simply a sample from the possible problem instances that AU can be confronted with. More in particular, the initial racing population consists of 10 decision trees with $cp \in \{2^{-1}, \ldots, 2^{-10}\}$ and the 10 regression subsets listed in Table 3.1 on page 53. The test cases are randomly drawn from the set of 24 instance classes of our concern and instantiated with a tardiness factor and range of due dates drawn randomly from a uniform distribution between 0 and 1.

Figure 3.4 gives a series of box–and–whisker plots on the basis of five such races. For each of the 20 candidates, the black dot indicates the median number of test cases that the candidates survived before being discarded. The race experiments started discarding after five test cases and tested the candidates on at most one hundred test cases. Hence, from the figure you can conclude that decision trees with cp set to 2^{-9} and 2^{-10} were never discarded. Note also how the number of test cases that decision tree models survive the race increases with the complexity of the tree. In comparison, the relative performance of linear models is much more erratic. Last, the width of the boxes indicates that there is quite a lot of variance among the five races. This variance among the races is probably due to the fact that all instance classes are lumped together in these races and that the models yield quite different relative performance on distinct instances.

As for model selection, the races unanimously chose the tenth decision tree model as the best one and so this model is used to determine the k values for apparent urgency in the rest of this thesis.

One final remark: The fact that the races show an overall preference for decision tree models over linear models does not imply that decision trees are superior in general. It only indicates that the given set of decision tree models is preferred over the given set of linear models. If smaller trees would have competed against

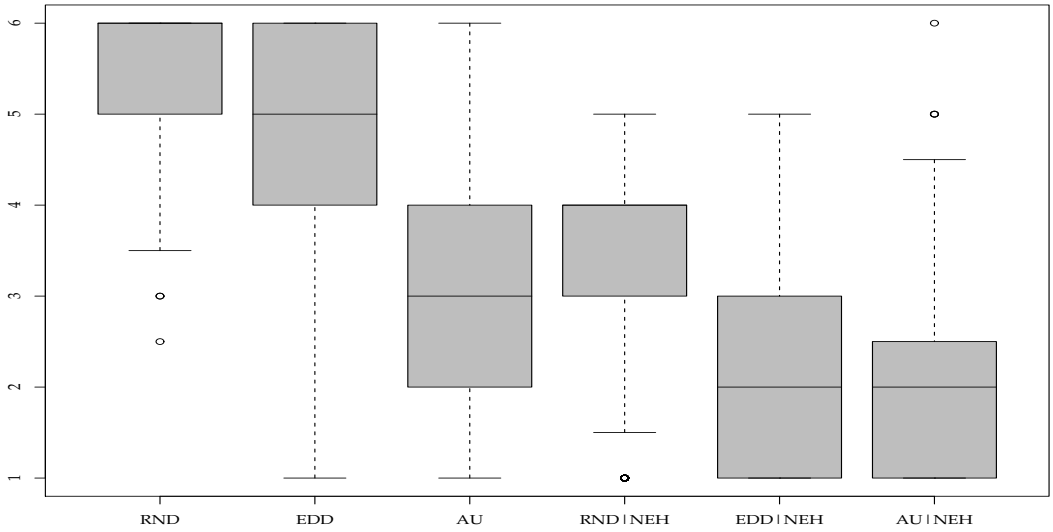


Figure 3.5: Ranks of construction heuristics.

larger linear models a completely different outcome could have ensued. Then again, perhaps a non–linear approximation is simply better.

3.2 Construction Procedures

The dispatching rules discussed in the previous section generate solutions to problem instances without much computational effort. However, it is often possible to construct solutions of considerably higher quality with some more computational effort. The construction procedure we consider here in addition to the dispatching rules is an insertion heuristic which we will refer to as NEH after [Nawa 83, Kim 93a] who were the first to apply it in the flow shop environment. *NEH* takes a stack of jobs and repeatedly pops the top job from the stack in order to insert it in the solution it is building until the stack is empty and the solution is complete. The insertion position in the partial solution is determined by evaluating all candidates partial solutions against the objective function and choosing the insertion point where the resulting partial solution has minimum objective function value.

The effectiveness of NEH depends on the order in which it considers the jobs for insertion. In order to find out how the order of jobs affects NEH, the following experiment was carried out: Generate a random instance of any of the problems; generate a job order in three different ways and evaluate the solutions corresponding to the job orders; finally, generate a solution with NEH using each job order as input; repeat the preceding procedure a sufficient number of times. The experiment yields six vectors of solution values whose length equals the number of repetitions, which turned out to be 4490 after one night running — long enough

to ensure that the whole problem domain has been covered. The six matching algorithmic variants are RND, EDD, AU, RND|NEH, EDD|NEH, and AU|NEH.

RND Solution consists of a random order of jobs.

EDD Solution consists of jobs sorted in increasing due date order.

AU Solution consists of jobs sorted according to the apparent urgency dispatching rule whose look-ahead parameter was deterimined with the largest decision tree generated in the previous section.

***|NEH** Solution is constructed with an insertion method that considers jobs in the order specified by the *.

Figure 3.5 on the page before compares the ranks of the six variants. From the figure it is clear that the extra computational effort of NEH results in better solutions most of the time. Moreover, the order in which NEH considers the jobs matters. Yet, it does not seem to matter greatly whether the jobs are in AU order or in EDD order although the AU order in itself is clearly much better than the EDD order.

A more detailed picture of the relative strengths of the six variants is given in Figure 3.6 on the facing page. The figure pictures for each combination of instance class and tardiness factor and due date which method performs best. In case of a draw, one of the winning variants is picked at random and depicted. Here are some observations:

- In flow shop problems, particularly those where there are no weights involved, the order in which NEH picks the job does not matter. For, in these areas the points representing AU|NEH, EDD|NEH, and RND|NEH appear with almost equal frequency.
- AU|NEH is a strong combination in particular when the range of due dates is not too large when the tardiness factor is not extreme. That is, the + character representing AU|NEH is especially dominant in the mid range of the left half of the plots.
- NEH is relatively weak when the tardiness factor is small and the range of due dates large. For, in the lower right corner of the plots, points representing non-NEH variants appear quite often.
- EDD|NEH is a strong combination when the tardiness factor is large and the range of due dates is large. That is, the black blocks representing EDD|NEH are dominant in the upper right corner of the plots.

Figure 3.7 on page 60 shows a decision tree that has been generated on the basis of the same set of data. The tree predicts which combination of dispatching rule and insertion heuristic is most likely to come out as the best based on the experience represented by the data. As one would expect the tree reflects Figure 3.6 in that either AU|NEH or EDD|NEH are selected — AU|NEH for parallel

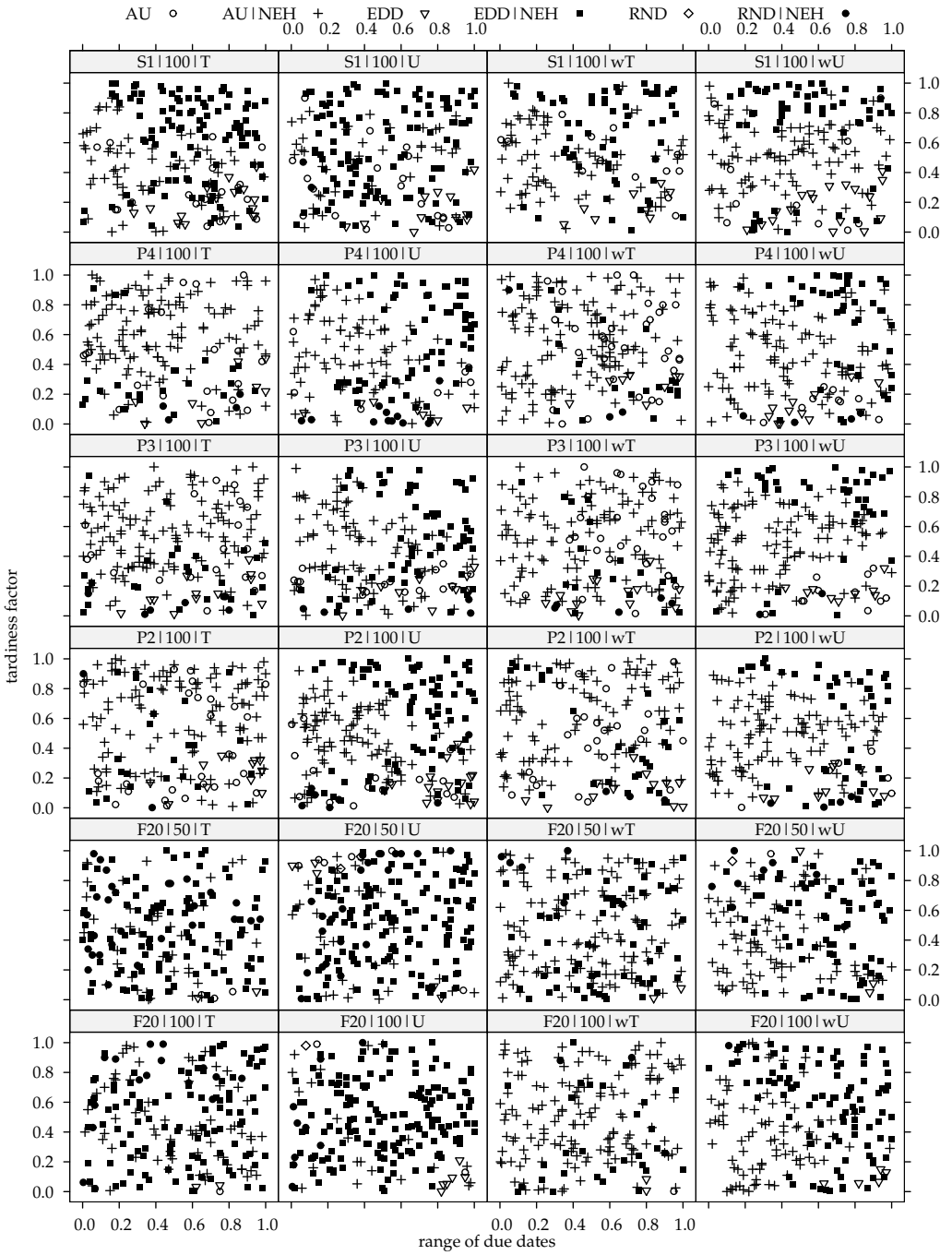


Figure 3.6: Winning construction heuristic for the tested combinations of problems with tardiness factor and range of due dates.

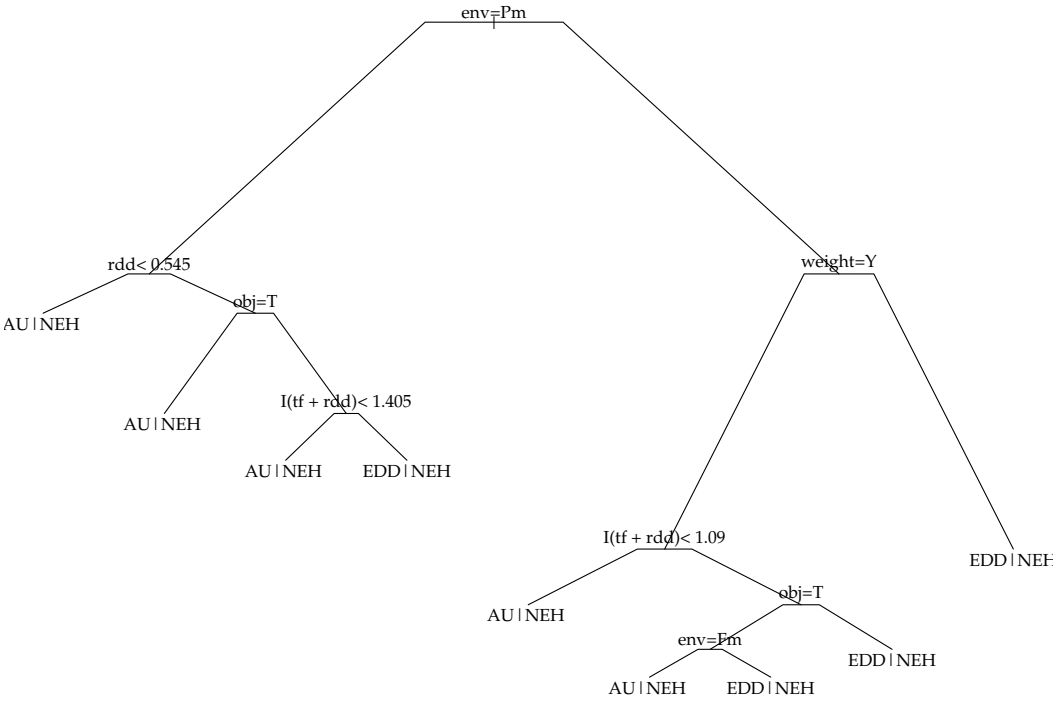


Figure 3.7: Decision tree for the selection of construction heuristics. Splitting criteria check whether the instance is drawn from a parallel machine problem (env=Pm); if yes, whether the range of due dates (rdd) is small enough; and whether the objective function measure tardiness or unit penalty (obj=T); if no, whether jobs are weighted and whether the sum of tardiness factor and range of due dates exceeds a threshold.

machine problems except in cases where $RDD > 0.5$, a unit penalty is applied, and $TF + RDD > 1.4$; EDD|NEH in other environments if no weights are involved or if $TF + RDD > 1.1$ and the problem in question is not $Fm|\text{prmu}|\sum T_j$.

A possible interpretation for the picture emerging from Figures 3.7 and 3.6 is the following. In flow shop problems, dispatching rules may have little effect because the machine environment is too complex to be represented by the simple scoring functions used in the dispatching rules. In problem instances with a small tardiness factor, the average due date is close to the makespan of the optimal solution and so most jobs will tend to complete before their due date, which makes these instances easy to solve. Therefore, NEH is redundant here. In contrast, when the tardiness factor is large and when the range of due dates is large as well, then there are many different solutions and just a few of them are very good. It may be that the relative weakness of EDD prevents NEH from converging towards a local optimum too soon.

3.3 Summary

This chapter described the implementation and calibration of algorithms for the generation of initial solutions that can be used to bootstrap the search for better solutions described in subsequent chapters. Two classes of techniques for the generation of solutions from scratch were discussed. The first class encompasses heuristics for ordering of jobs according to their due dates, weights, or processing times. The second class encompasses techniques for the construction of solutions through iterative insertion of jobs in a partial sequence. Particular attention was paid to the calibration of apparent urgency, the most promising heuristic from the first class. A generic implementation of this heuristic for all scheduling problems of interest was proposed and extensive experiments to guide the calibration of the heuristic to each specific problem were recounted: For every instance class, a racing experiment was performed to determine the best parameter setting; models were then developed that related the best parameter setting to features of the instance classes; and finally a racing experiment was carried out to select the best model out of the multitude that had been developed.

For the second class, the insertion mechanism is the generic element and the order in which jobs are considered for insertion is problem specific. It turns out that the job order that in itself yields the highest solution quality is also the best order in which jobs should be considered for insertion. Moreover, the addition of the extra insertion filter leads to an significant inprovement in overall solution quality. Therefore, the initialisation procedure that comes out of this chapter is to first order jobs according to a version of apparent urgency that is calibrated with respect to problem characteristics; the order thus found is the order in which jobs are then considered for insertion in the partial solution, where the partial solution is the sequence of jobs inserted so far and insertion is allowed everywhere on that sequence. When all jobs have been inserted, the bootstrap is complete.

4 Local Search

In the previous chapter, we saw how to construct a solution from scratch. In this chapter, we will review some simple methods to improve an existing solution by means of small modifications. Two methods in particular are explored. The first is iterative improvement, the most straightforward and most common variant of local search. The second is piped iterative improvement which is a variation on variable neighborhood descent and a simple, but powerful, extension to local search.

4.1 Iterative Improvement

Iterative improvement is local search stripped bare from the bells and whistles that come with metaheuristics like simulated annealing, tabu search, or variable neighborhood search. Even so, there are still quite a few decisions to be made by the developer who decides to implement iterative improvement and some testing is needed in order to get a feel for the effect of the design choices and to weed out variants of iterative improvement that perform below par. The description and analysis of these tests is the main thrust of this section. But first we review the design and implementation of local search.

Framework

Like any other type of local search, iterative improvement algorithms start at some location of a given search space and subsequently move from the present location to a neighboring location, where each location has only a relatively small number of neighbors and where each of the moves is determined by a decision based on local knowledge only [Hoos 04]. In the design of any iterative improvement, one has to decide on solution representation, search neighborhood, and pivoting rule.

Steepest descent, which is presented in Figure 4.1 on the following page, is a classic exemplar of local search. It is also known as *best improvement local search* and is a variant of iterative improvement. The algorithm is called steepest descent because the neighboring solution that has the highest quality is selected in each step. Other pivoting rules are also possible. For instance, in *first improvement* local search, the first neighbor in the neighborhood that is found to yield an improvement is selected.

The picture to the left of the outline in Figure 4.1 illustrates the behavior of local search. The dots represent the search space, the numbers next to the dots correspond to the quality of the solutions represented by the dots, and the dashed lines

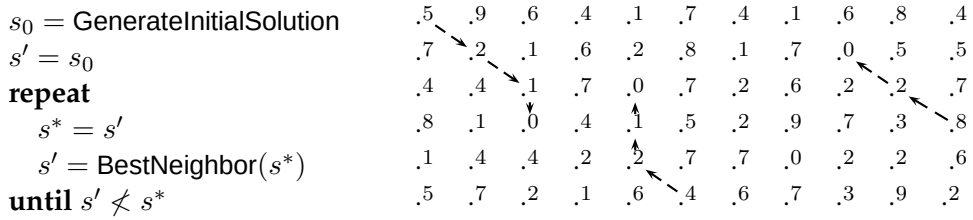


Figure 4.1: First improvement local search: Outline & illustration.

delineate the paths that a local search might traverse through the search space. Each series of arrows represents a separate path. In this particular instance, the neighborhood of a solution is defined as the set of dots that are adjacent to the dot that represents that solution. So, if the local search starts at the dot in the upper left corner, the algorithm has a choice between a solution with cost 7, a solution with cost 9, and a solution with cost 2. Since the algorithm is trying to minimize the cost, selects the solution with cost 2. For the next step, the algorithm has the choice between 8 neighbors. From the two neighbors with cost 1, the algorithm picks the one in the lower right corner of the neighborhood. This is a lucky choice, for one of the neighbors of this solution turns out the be a solution with the globally optimal cost of 0.

The picture in Figure 4.1 illustrates a number of facts about local search. First of all, the picture shows that there can be several solutions with the optimal quality as more than one point in the picture has a cost of zero. Secondly, the pictures shows that the global optimum cannot always be reached with just descent. For, if a local search were to start in the lower left corner and was allowed to move either horizontally or vertically but not along the diagonal, then the best solution it would find would have a value of one. Finally, the picture shows that the definition of the neighborhood affects the effectiveness of descent: If the local search that started in the lower left corner would have been allowed to move along the diagonal, it would have been able to reach the solution with zero cost.

Instantiation

In the local search implementations we consider in this chapter, a solution is represented by a permutation of jobs and the search neighborhood consists of all shifts and swaps of jobs that involve two positions in the permutation. The pivoting rule comes in two flavors: Best improvement and first improvement. *First improvement* local search immediately executes an improving modification to the solution when it finds one; *best improvement* local search first evaluates all modifications in the search neighborhoods and then executes the best one. Note also, that both improvement methods scan the neighborhoods in the same fashion in that both first consider jobs at the beginning of the permutation and then gradually work their way upwards. Both best and first improvement local search return to the starting position after each executing of an improvement on the permutatation.

As for the neighborhoods, they are defined by the types of modifications or

moves that the algorithm considers. In addition, local search neighborhoods are constrained by problem specific circumstances that may induce the algorithm to ignore certain potential modifications to the solution, for example because it is known that these modifications cannot possibly lead to an improvement in solution quality. In our case, we consider three types of modifications: a swap of two jobs, a shift backwards of one job, and a shift forwards of one job. With these three moves, we can construct seven distinct neighborhoods: There are three neighborhoods that consider only one type of moves. In addition, there are three neighborhoods that consider two types of moves and pick whichever yields the best improvement. Finally, there is one neighborhood that considers all three types of moves.

One solution representation, two pivoting rules, and seven neighborhoods add up to a total of fourteen local search configurations to be considered. Each is identified by an acronym which is made up of a capital letter *B* or *F* indicating whether a best or first improvement pivoting rule is employed, and one or more lowercase letters for each type of move — *b* for backward shift, *f* for forward shift, and *s* for swap — that is considered. Thus, we have *Fb, Ff, Fs, Bb, Bf, Bs, Fbf, Fbs, Ffs, Bbf, Bbs, Bfs, Fbfs*, and *Bbfs*.

Data Generation

The local search algorithms were written in C++. More in particular, the evaluation techniques described in Section 1.4 were implemented and embedded in the local search framework described above and in Chapter 1. The code was compiled with `gcc` with the optimization flag set to O3. All algorithms were then tested on a computer running Linux.

In order to test the local search, several instances were randomly generated according to a problem specification that was randomly draw from the twenty–four problem specifications availble and with a tardiness factor and range of due dates randomly drawn from a uniform distribution between zero and one. All fourteen variants were tested on each instances that was generated and for each pair of problem instance and iterative improvement variant the quality of the solution and the run time required were stored. Finally, R's scale function is applied to the data to allow for comparisons among the instances so that the eventual scaled values show the deviation from the average performance on an instance after normalization with respect to the variance in distribution of performances on that instance.

Observations

Figure 4.2 on the next page and Figure 4.3 on page 67 are scatter plots comparing the relative run times and solution qualities of local search variants differentiaed by idxneighborhood!in iterative improvement The effect of the choice of a pivoting rule can be deduced from a comparison between Figure 4.2 and Figure 4.3. What the comparison reveals is that the first improvement variants tend to require less time to complete the search than their best improvement variants and nonetheless

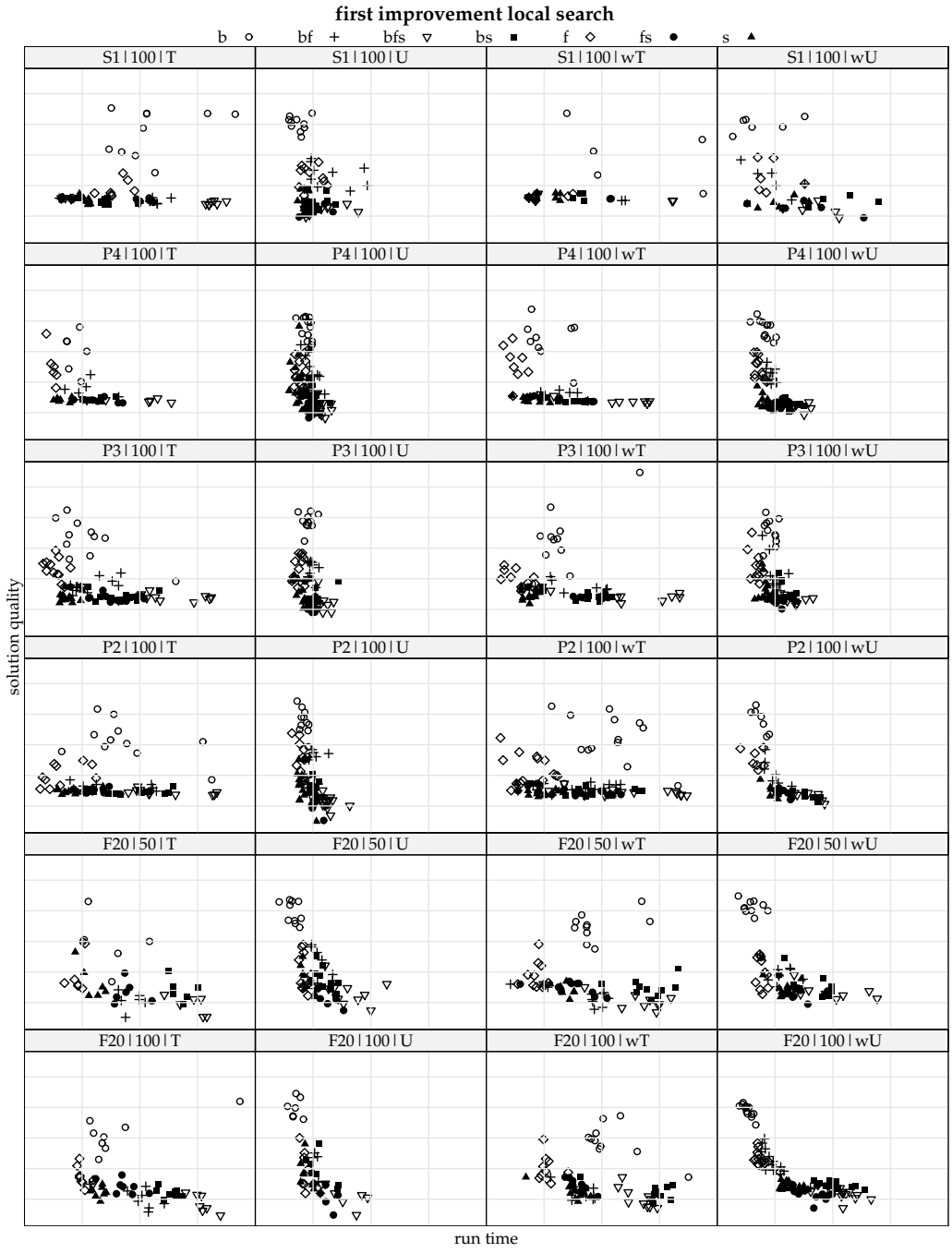


Figure 4.2: Iterative improvement run time versus solution quality (I).

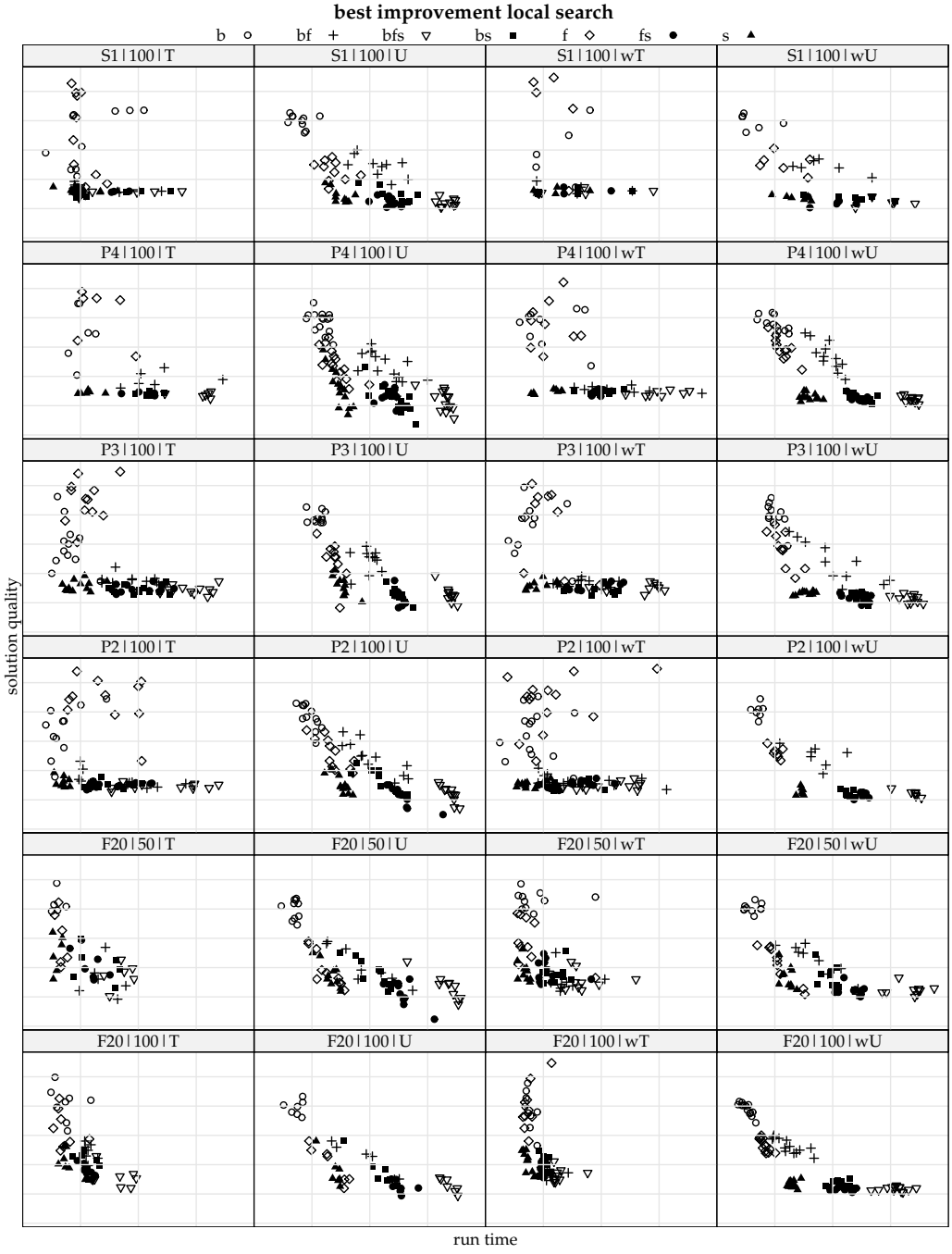


Figure 4.3: Iterative improvement run time versus solution quality (II).

manage to obtain a similar range of solution qualities. The effect is particularly pronounced for the problems where the objective is to minimize the number of tardy jobs (in the second column) and becomes less so as the complexity of the objective function increases. In fact, for the instances of $Fm|\text{prmu}|\sum w_j T_j$ (the last two plots in column three), it might even be argued that the effect is partially reversed because in these cases best improvement variants have a shorter run time than the first improvement variants while the latter still manage to obtain a better score on the objective function than the former.

If we now turn our attention to the neighborhoods again, Figure 4.2 and Figure 4.3 tell us that neighborhood size has a negative effect on run time and a positive on solution quality and that the swap move is more effective than the shift moves. The combination of backward shift and forward shift performs better either move taken separately but still does not attain the performance of the neighborhood based on swap alone most of the time. Yet, the addition of either backwards shift or forward shift or both to a swap neighborhood does often yield solutions of higher quality, be it at the expense of extra run time.

Classification

Complementary to the scatter plots of Figure 4.2 and 4.3, we can also use the data to try and detect similarities in the performance of the variants.

Figure 4.4 on the next page gives a clustering of the fourteen local search variants. The first tree from above is the hierarchical cluster obtained by running R's hclust method on the scaled solution quality data and the other tree is the hierarchical cluster obtained by the same method on the run time data. The cluster are based on the performance of the fourteen variants on 600 randomly generated instances.

In the cluster based on solution quality, local search variants with neighborhoods that contain a backward move are often next to variants with neighborhoods that are similar except for the omission of the backward move. Hence, we can conclude that the addition of the backward shift to a neighborhood has little effect on the solution quality yielded by the resulting algorithm. Futhermore, the first and best improvement version of local search on a particular neighborhood are often clustered together. So the choice of pivoting rule does not affect solution quality either. In the cluster based on run time, first improvement variants are rather neatly separated from best improvement variants and so here we can conclude that the choice of pivoting rule *does* affect the run time. A distinction that is even higher up in the hierarchy is between neighborhoods that contain only one move type and neighborhoods that contain more than one type. Apparently, neighborhood size affects run time even more. Figure 4.5 on the facing page is a classification tree that was obtained by feeding the same run time and solution quality data along with a description of the instances to the rpart hierarchical partitioning method in R. It shows it is possible to classify the fourteen variants on the basis of run time and solution quality alone.

In the tree, the labels of the leafs indicate which variant is most likely to fit

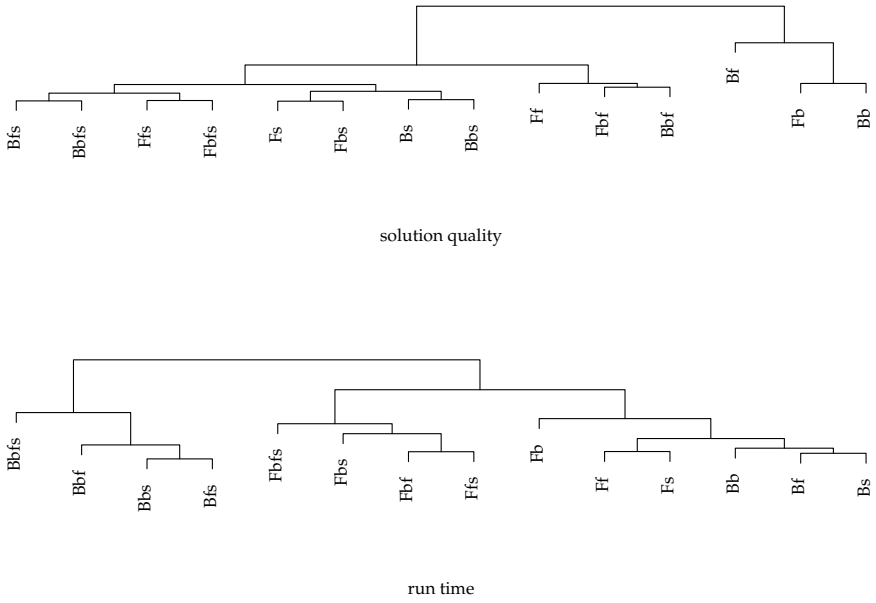


Figure 4.4: Hierarchical clustering of iterative improvement variants.

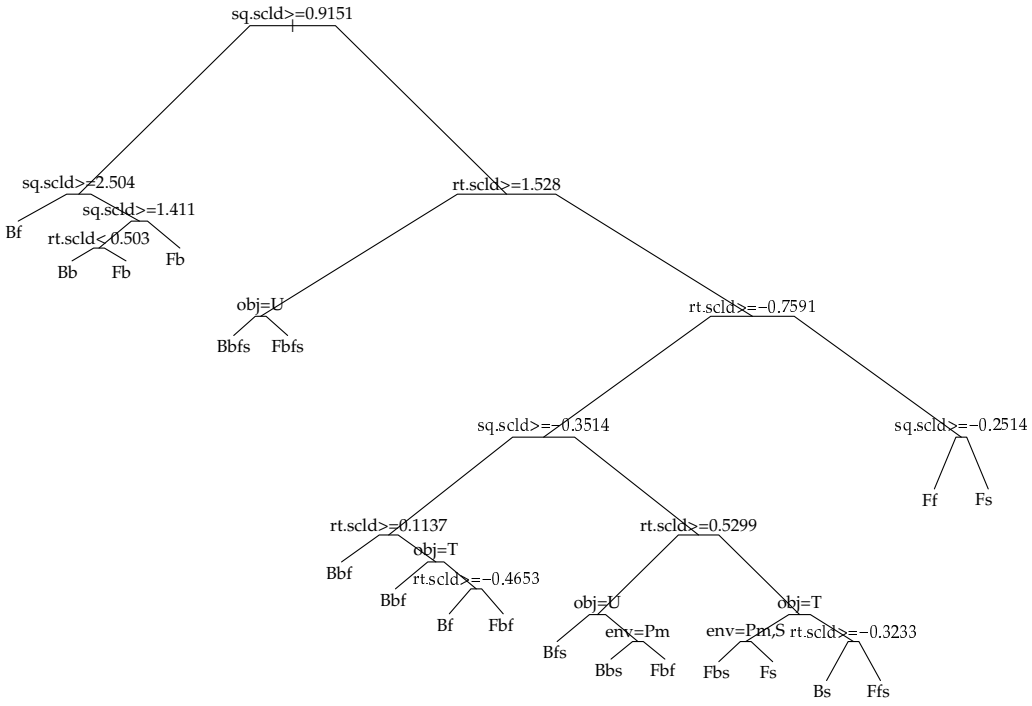


Figure 4.5: Hierarchical identification of iterative improvement variants.

the description in the path from root to leaf. The splitting criteria are mostly related to solution quality and run time. Therefore, it seems safe to infer that the trade–off between run time and solution quality attained by the variants of iterated improvement that were tested is relatively independent from issues of problem specification. The tree corroborates the impressions from the scatter plots in that *Bf*, *Bb* and *Fb* are best described by a low solution quality and hence, since we are attacking a minimization problem, a high value of *sq.scld*. At the other end of the spectrum, *Bbfs* and *Fbfs* feature above average solution qualities and even more extreme run times. In between, variants based on the swap move with the occasional extension of a backward of forward shift emerge with *Bbf* and *Fbf* as close contestants.

Interpretation

How can the differences among the local search algorithms be explained? Part of it may be due to implementation details. Recall that in the parallel machine environment modifications on each of the single machines are tried before considering the moves of jobs between machines. This might explain why the differences in run times between first improvement and best improvement become more pronounced as the number of machines increases. For, the evaluation of moves within a machine is far less expensive than the evaluation of moves between machines and first improvement will be more likely to consider only the former, whereas best improvement has to consider all. Also the difference between first improvement and best improvement in the single machine environment may be due to the implementation: The evaluation speed–up techniques and the neighborhood reduction described in Section 1.4 have a greater effect in combination with a best improvement pivoting rule since the whole neighborhood rather than a part needs to be scanned each iteration according to this rule. And, most importantly, the effect of the choice of pivoting rule is bigger for total (weighted) tardiness problems because the speed–up gains per evaluation are more pronounced relative to the speed–up gains that could be achieved in the computation of total unit penalty.

Also artifacts of instance generation may play a role. What I have in mind is the detected effect of the interaction between tardiness factor and machine environment. Tardiness factor is defined relative to the optimal makespan of the jobs in the instance. In single machine instances, the makespan is simply the sum of all processing times; for parallel machine instances, the makespan is approximated by dividing the total processing time by the number of machines in the environment; for flow shop instances, makespan is known because the instances are derived from a standard benchmark set for which makespans have been computed. It could well be that these differences in definition have some bearing on the final data.

However, apart from the effects of artifacts and implementation, some of the algorithm behavior must be due to pure problem specification. For, how else could it be that also in the case of flow shop environment where hardly any speed–up tricks were applied, first improvement and best improvement local search exhibit

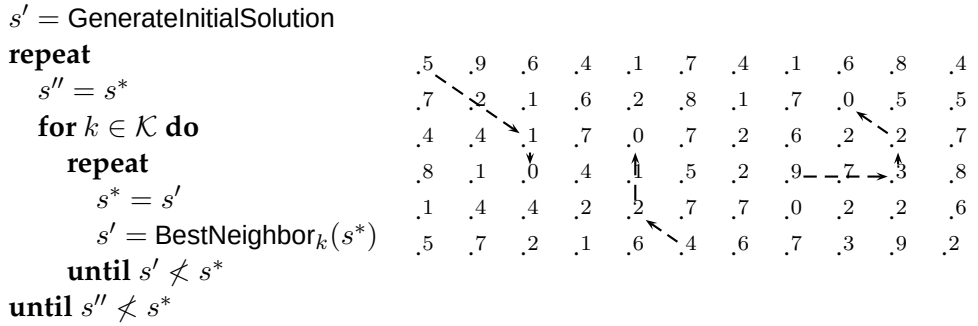


Figure 4.6: Piped iterative improvement: Outline & illustration.

similar behavior as in the other environments? Also in the flow shop environment, first improvement is relatively slow for total tardiness problems and relatively fast for unit penalty problems. Why do certain neighborhoods provide a better performance for certain instance classes and for certain pairs of tardiness factor and range of due dates — and not for others?

Let's hope that future research can shed a light on these issues. For now, it is time to move on to the next step in the development process.

4.2 Piped Iterative Improvement

This section investigates what happens if we put a selection of the iterative improvement variants of the previous section after one another. After a recapitulation of the virtues of sequencing or piping, a description is given of the candidates of piped iterative improvement that are subjected to tests and the design of the experiments is specified. Finally, an analysis of the experimental results is performed to link the composition of piped local search to performance on the various scheduling problems under consideration.

Framework

Piped iterative improvement (PII) is nothing more and nothing less than the search through a series of neighborhoods in sequence. The outline for this procedure is given in Figure 4.6. First, PII performs an iterative improvement local searh on one neighborhood. If no improvement is found, it moves on to the next neighborhood and it keeps on iterating through the neighborhoods until all neighborhood search potential is exhausted. The implementation of PII need not be terribly complicated in that one can simply take the existing implementations of iterative improvement and redirect the output of one variant so that it can serve as input to another variant.

PII algorithms are named according to the following scheme: $P\mathcal{N}_1|\ldots|\mathcal{N}_n$. In this scheme P is one of B and F and indicates whether search is done according to the first improvement or according to the best improvement pivoting rule, $\mathcal{N}$ is a neighborhood defined in the same way as in the previous section, and bars (|) are

used to separate the neighborhoods.[1] The order of the neighborhoods corresponds to the order in which the neighborhoods are considered by PII.

To see why PII might improve upon iterative improvement, consider the picture inFigure 4.6. The search space is depicted as a matrix of dots and the value next to each dot indicates the quality or cost associated with the dots. The arrows denote the three paths that PII would follow if it started either from the upper left corner, from the sixth position on the last row, or from the eighth position on the fourth row. The variant of PII that would walk the paths depicted on the matrix is *Bh|v|d*. That is, first horizontal moves are thried, then vertical moves, and finally diagonal moves. So, if *Bh|v|d* were to start in the upper left corner, it would first try horizontal moves and then vertical moves, but find no improvement. The neighborhood of diagonal moves would allow it to proceed a while until a local optimum was reached at the third cel of the third row. Then, *Bh|v|d* would revert to horizontal moves, move on to vertical moves and find the global optimum in the third cel of the fourth row. A similar story can be told for the other starting positions. Each time, PII is able to reach parts that the three iterative improvement procedures cannot reach on their own.

Experimental Setup

In principle, one could take all fourteen iterative improvement from the previous section and try to find a subset and order of these variants. The problem is that there are quite a few ways to put (a subset of) fourteen elements in order, $\sum_{i=1}^{14} \binom{14}{i} i! = 236975164804$ to be precise. And so we have to confine our search. The first restriction that suggests itself is to consider to fix the pivoting rule for all local search components. This makes sense because extensive research suggests that the combination of a variety of neighborhoods and not so much the method used to scan the neighborhoods is what make the sequencing of local searches work [Hans 03]. In this way, we are able to reduce the search space to a total of $2 \times \sum_{i=1}^{7} \binom{7}{i} i! = 27398$ variants. But 27398 is still a rather large number and so we need a second restriction. This second and last restriction is to limit the number of different neighborhoods to be considered by a single PII variant to three. Since there are only three distinct types of neighborhoods moves, three neighborhoods suffice to represent each of the types separate or in combination. Thus, the total number of variants is reduced to $2 \times \sum_{i=1}^{3} \binom{7}{i} i! = 518$, which is reasonably small.

In practice, it turned out to be rather cumbersome to enumerate all 518 variant. So, instead, an enumeration was done of all combinations of the seven neighborhoods plus an empty neighborhood on each of the three positions in the neighborhood order. That enumeration yields $2 \cdot 8^3 = 1024$ of which 518 are unique. The redundancy occurs because $b|b|b \equiv b|| \equiv |b|$ and $f|b|b \equiv f||b$ and so on and so forth. This redundancy is not a big problem, as selection procedures like racing will still be able to function. However, racing will become slower as a result because more variants have to be tested at the beginning of the race and possibly because the rate with which underperforming candidates are discarded is lower. Furthermore, the

[1] In Unix shells, the | symbol is used to contruct a pipe between programs; that's why.

Table 4.1: Properties of race–survivors: Number of survivors out of 1024 (#); Proportion using best improvement (B, 0.5 initially); most common move type (m); Average number of neighborhoods (N, $2\frac{5}{8}$ initially); Average number of move types in the neighborhoods ($|m|$, 1.71 initially).

	#	B	m	N	$\|m\|$		#	B	m	N	$\|m\|$
$1\|\|\sum_{j=1}^{100} U_j$	5	0.00	s	1.2	1.3	$P4\|\|\sum_{j=1}^{100} U_j$	4	0.25	s	1.0	1.2
$1\|\|\sum_{j=1}^{100} T_j$	9	0.22	s	1.7	1.1	$P4\|\|\sum_{j=1}^{100} T_j$	19	0.47	f	2.3	1.1
$1\|\|\sum_{j=1}^{100} w_j U_j$	18	0.22	f	1.9	1.2	$P4\|\|\sum_{j=1}^{100} w_j U_j$	9	0.11	f	2.3	1.2
$1\|\|\sum_{j=1}^{100} w_j T_j$	15	0.53	f	2.2	1.2	$P4\|\|\sum_{j=1}^{100} w_j T_j$	25	0.48	f	2.2	1.2
$P2\|\|\sum_{j=1}^{100} U_j$	5	0.00	f	1.8	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{50} U_j$	5	0.00	s	1.0	1.0
$P2\|\|\sum_{j=1}^{100} T_j$	20	0.75	b	2.3	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{50} T_j$	65	0.51	f	2.3	1.3
$P2\|\|\sum_{j=1}^{100} w_j U_j$	9	0.22	f	1.9	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j U_j$	12	0.33	f	1.5	1.2
$P2\|\|\sum_{j=1}^{100} w_j T_j$	39	0.62	f	2.2	1.2	$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j T_j$	76	0.55	f	2.3	1.4
$P3\|\|\sum_{j=1}^{100} U_j$	4	0.25	f	1.8	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{100} U_j$	6	0.00	s	1.2	1.0
$P3\|\|\sum_{j=1}^{100} T_j$	20	0.65	f	2.0	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{100} T_j$	133	0.65	f	2.3	1.5
$P3\|\|\sum_{j=1}^{100} w_j U_j$	11	0.18	f	1.8	1.1	$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j U_j$	18	0.33	f	1.9	1.2
$P3\|\|\sum_{j=1}^{100} w_j T_j$	34	0.44	f	2.2	1.2	$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j T_j$	91	0.47	s	2.4	1.5

set that survives the races needs some post–processing so that the analysis of the results is not clouded by the redundancies.

Racing constituted the main experiment that was carried out in order to gain insight in the behavior of PII. The races, one per instance class, were configured to select the non–dominated set of algorithms that takes into account both run times and solution qualities along the lines delineated in Section 2.2 on page 32. Each race was run with up to 100 randomly generated test instances and a maximum number of experiments of 10×1024. The minimum number of test cases per candidate before discarding was set to five.

Results

The final selection of the races is listed in Appendix A.2. For each instance class, the appendix provides a list of the surviving PII candidates ordered according to their average solution quality on the test instances. In addition, Table 4.1 gives a summary of the results.

The first column of Table 4.1 lists the number of variants that survived each race. Clearly, the reduction is considerable, especially with respect to the unit penalty objective function. The second column of Table 4.1 indicates for each problem class which proportion of the survivors employed the best improvement pivoting rule. What is shows is that, in particular when unit penalty problems are concerned, but also for many total tardiness problems, the first improvement pivoting rule is preferred over the best improvement variant. Furthermore, in the context of the parallel machine environment, there seems to be a slight positive correlation between the popularity of the first improvement pivoting rule and the number of machines employed. The third, fourth, and fifth columns of Table 4.1 are con-

cerned with the neighborhoods that survived. The characters in the third column indicate which type of move occurs most often in the surviving neighborhoods. Particularly peculiar is the contrast between the ubiquity of the forward shift and the near absence of the backward shift. The average number of neighborhoods in PII is given in the fourth column and the average number of distinct move types in the neighborhoods is given in the fifth column. Apparently, it is a good idea to search several neighborhoods in sequence and, apparently, in that case the complexity of the neighborhood should be small. This hunch is supported by the listing in the appendix, where the variants with one complex neighborhood tend to end in front of the row and the variants with one simple neighborhood tend to end at the end. The front and the end represent good solution quality with bad run time and good run time with bad solution quality respectively. The variants with multiple neighborhoods end up somewhere in between, which means they probably strike a better balance between run time and solution quality.

Analysis

The experiments can be analyzed from several angles. One can try to detect similarities between the races, one can look at the similarities between algorithms, and one can try to predict the race survival of algorithms.

Similarities among Races Figure 4.7 on the facing page is a hierarchical clustering of the survival times of the algorithms in the races. The cluster is based on the number of test cases that each candidate was able to run in each race before being discarded. Apparently, the shape of the objective function is important. All races concerned with the minimization of the total number of late jobs belong to one cluster. Furthermore, the number of jobs in flow shop problems does not seem to matter for the course of the associated race. Finally, in the flow shop problem with a total tardiness minimization objective function, it does not seem to matter very much whether the jobs are weighted or not, but for the other environments it mostly does.

Similarities among Candidates Cluster analysis is also useful to find out what variants of PII are similar to each other. Figure 4.8 on the next page shows a hierarchical cluster of a selection of variants based on the run times and solution qualities on the first five test cases in each of the 24 racing experiments. The selection consists of the top fifty most common survivors of the races after adjustment for obvious equivalences in specification. For instance, $Bb||$ and $B|b|$ are counted as one. Anyway, what is important about this figure is not so much the selection, but rather the demarcation lines between clusters of the selection that emerge.

In Figure 4.8, the clusters of PII variants are ordered in a hierarchical tree. At the root of the tree a first distinction is made between $Ff||$, $Ff|b|$, $Bf|b|$, and $Bb|f|$ on the one hand and the rest of the selection on the other hand. What distinguishes the latter group from the former group is that the variants in the latter group are more powerful variants of iterative improvement in that they tend to contain more and

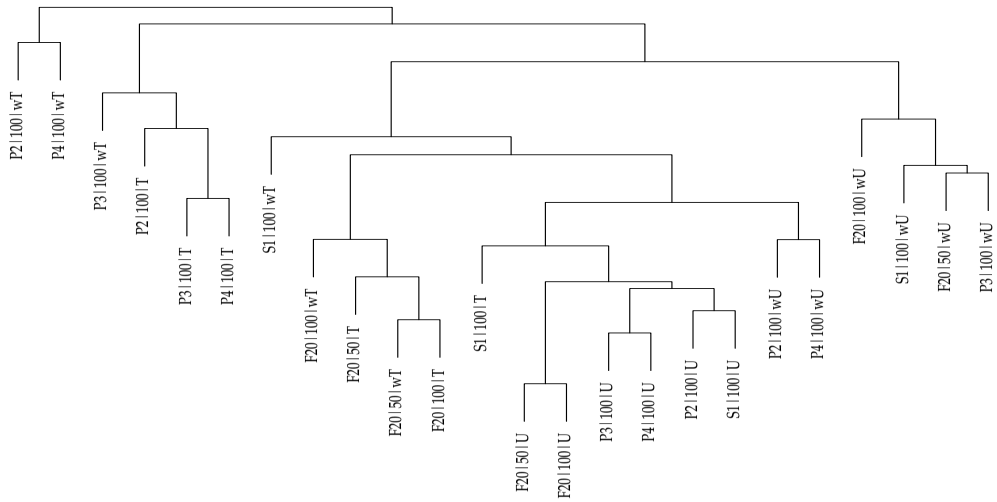


Figure 4.7: Similarities among races.

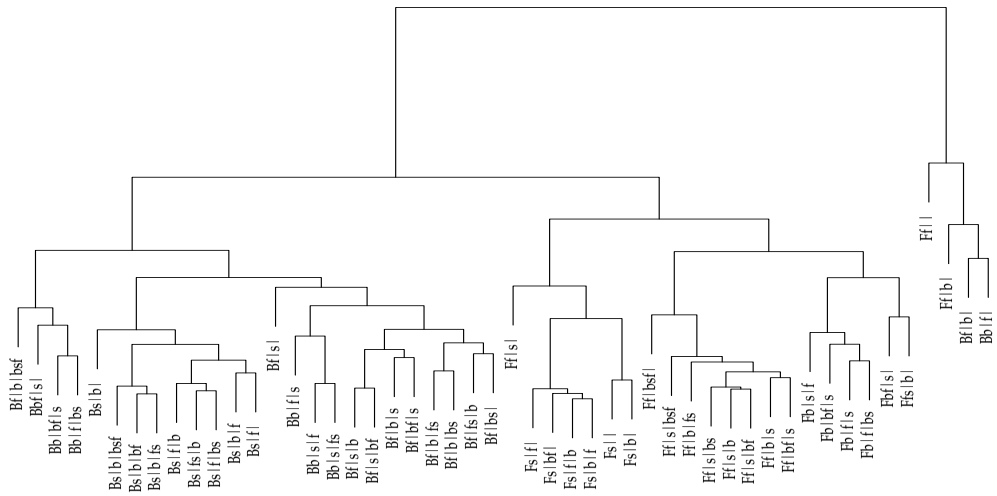


Figure 4.8: Similarities among candidates: Clustering of top 50 most selected piped iterative improvement variants.

bigger neighborhoods. The second level of distinction is between best improvement variants on the one hand and first improvement variants on the other hand. Apparently choice of pivoting rule matters. If we finally jump to the labels at the leaves of the tree, what stands out is how neatly the variants are ordered according to the order in which they consider the neighborhoods.

Survival Analysis For each of the races, it was tabulated if and when each of the candidates was discarded. These data were then used to construct a decision tree. Each race is uniquely identified by the problem objective, existence of weights, machine environment, number of jobs, and number of machines and each candidate is uniquely identified by the pivoting rule and the neighborhoods that it employs. In addition, the candidates are characterized by the number of non–empty neighborhoods that are searched. For instance, *Fb||* has one neighborhood and *Fb|bs|* has two neighborhoods. Also, the number of occurrences of certain move types is counted. For instance, *b*, the backward shift move, occurs once in *Fb||* and twice in *Fb|bs|*. Finally, the size of each neighborhood — that is, the number of distinct move types considered — is counted. For instance, *Fb|bs|* has one neighborhood of size one and two of size two and a combined neighborhood size of three. Now, the decision tree in Figure 4.9 on the facing page is what you get if you put all these elements together and use it to partition the candidates according to their likelihood of survival. The values on the leafs of the tree indicate the hazard level associated with the candidates. A low hazard level means a high chance of survival and a high hazard level means a low chance of survival.

From the decision tree in Figure 4.9, we can conclude that the total number of move types that are considered is the best overall predictor of a candidate's chance of survival. The fact that the cutoff point in the decision tree is at 3.5 is in agreement with the hunch that what is most important for PII is that each of the move types (*b*, *f*, and *s*) are considered but that there is little additional value in considering a move type in the current neighborhood that was already part of the previous neighborhood. In short, when more than one neighborhood is searched, the neighborhoods should be complementary.

Another thing that we can conclude on the basis of the decision tree is that candidates of like *Bb|f|s* are likely to survive because they are constructed of simple complementary neighborhoods. Candidates that employ more complex neighborhoods like *bs* can still survive, provided that the first neighborhood is simple. So the first neighborhood should lead PII to a good solution quickly, and the second and third neighborhood serve mainly to let PII go the extra mile.

The plots on the right of the figure shows how the combined neighborhood size of candidates affects their chances of survival in the races. The upper, middle and lower plot show the survival rates for PII variants consisting of one, two, or three iterative improvement components respectively. In each case, candidates with the smallest combined neighborhood size stand the best chance to survive and candidates with the largest combined neighborhood size stand the worst chance to

```
1:  if ∑|N_i| < 3.5 then
2:      if Tardiness Penalty then
3:          if |b| < 1.5 then
4:              if |s| < 1.5 then
5:                  if |f| < 1.5 then
6:                      if |N| ≥ 2.5 then
7:                          0.20
8:                      else
9:                          if Flow Shop then
10:                             0.45
11:                         else
12:                             0.71
13:                 else
14:                     0.92
15:             else
16:                 1.00
17:         else
18:             1.01
19:     else
20:         0.97
21: else
22:     if Tardiness Penalty then
23:         if |N_1| < 1.5 then
24:             if Flow Shop then
25:                 if N_2 ∉ {bf, bsf} then
26:                     0.59
27:                 else
28:                     0.95
29:             else
30:                 0.96
31:         else
32:             1.09
33:     else
34:         1.15
```

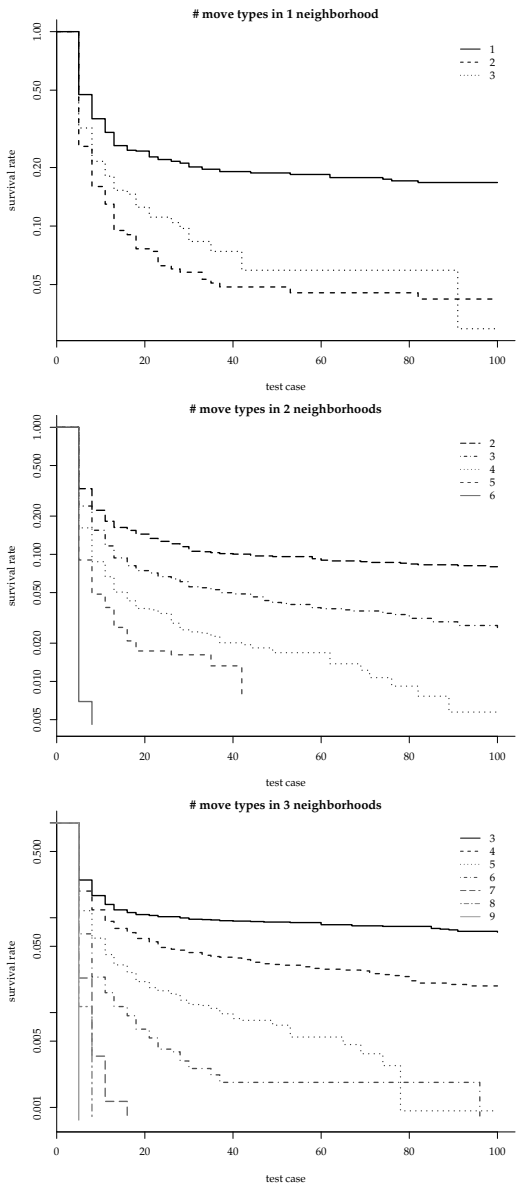


Figure 4.9: Survival analysis: The pseudo code on the left represents a decision tree for survival. Here, a low number in the leaf means a high chance of survival for the class delineated by that leaf. The plots on the right depict how the combined sum of the neighborhood ($\sum |N_j|$) affect the survival chance of PII. The plot above shows the survival lines for PII composed of one iterative improvement; the plot in the middle is for PII composed of two iterative improvement procedures; and the plot below is for PII composed of three variants.

survive. A comparison of the plots corroborates that it is better to concatenate various lean neighborhoods than to combine various moves in one big neighborhood: The PII variants of two iterative improvement components with each a neighborhood made up of just one move type in the middle plot survive the race more than 10% of the time. In contrast, the PII variants consisting of one iterative improvement with a neighborhood made up of two or more move types, survive the races less than 5% of the time. Similarly, the PII variants with three search components survive around 10% of the time when each neighborhood consists of only one neighborhood and hardly survive if the combined neighborhood size exceeds four.

The decision tree in Figure 4.9 is most specific about problems that feature an objective function that measures the total (weighted) tardiness of jobs. However, this does not entail that the above observations are only valid for the total tardiness objective in the flow shop environment. Rather, the rate of selection in the unit penalty single and parallel machine environments is so rigorous, that there are insufficient data available to make the same subtle distinction as were made in the case of total tardiness flow shop environments. Figure 4.10 on the next page gives the decision tree based on unit penalty problems alone. Again it shows that the combined neighborhood size is a crucial factor for the survival of the races. In contrast to the tardiness penalty problems, for the races on unit penalty problems, the choice of pivoting rule is important and the choice of move types matters less. Best improvement local search scans the complete neighborhood at each step; first improvement local search only scans up to the first improvement. Apparently, the resulting difference in computational effort starts to count when local searches are sequenced. A possible explanation for this is that the subsequent local search runs will start from good solutions and need only a few steps to reach the next local optimum. If the initial solution is bad, best improvement local search manages to compete with first improvement local search because it can find in a few costly steps what first improvement local search will find in many cheap steps. But, if the initial solution has a high quality, there will be few moves left that yield an improvement and so first improvement local search will not make many more steps than best improvement local search. This is especially important in parallel machine environments since scanning the whole neighborhood is potentially very expensive for these problems (see Section 1.4).

4.3 Summary

This chapter documented the use of racing for the development of local search and piped local search algorithms for deterministic scheduling models subject to tardiness penalties. First, 14 local search variants were tested on a grand total of 605 randomly generated problem instances. These preliminary results provided a basis to select 518 candidate configurations of piped local search. Racing was then applied to select the best configurations for each specific problem class. It

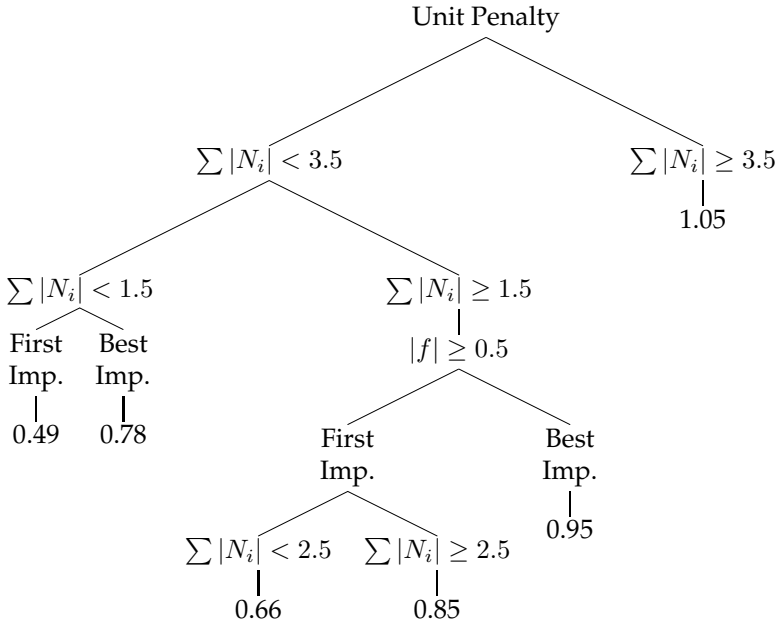


Figure 4.10: Decision tree for survival in unit penalty races.

was found that for local search the way in which the neighborhood is scanned as defined by the pivoting rule does not affect the overall quality of the solutions that are generated. However, the choice of the pivoting rule may impact the run time of the algorithm. Furthermore, it was found that solution quality and run time are correlated with the size of the neighborhood that is searched. Small neighborhoods yield low quality solutions within a short time. Large neighborhoods yield high quality solutions after a long time. With respect to piped local search, it was found that the combination of several neighborhoods is a good thing as long as the neighborhoods complement each other. That is, an extra neighborhood is useful only when it contains moves that were not possible in the neighborhoods that have already been searched. Furthermore, it was found that it is better to search through a series of small complementary neighborhoods than to search through one neighborhood in which the small neighborhoods are merged.

5 Iterated Local Search

This chapter describes the experiments that were done in order to find instantiations of iterated local search (ILS) that perform well on the scheduling problems with tardiness penalties described in Chapter 1. First, there is a recapitulation of iterated local search and a review of its application to combinatiorial optimization (Section 5.1). Next, there is a description of the experiments and a discussion on the interpretation of the results (Section 5.2). In addition, there is a discussion on commonalities among problems and ILS candidates and there is a small investigation into the robustness of the results. Finally, there is a summary (Section 5.4).

5.1 Framework & Instantiation

Iterated local search [Lour 02] is an extension to local search. Local search, in particular iterated improvement, has the unfortunate habit of stopping before finding the optimal solution to the problem that it is supposed to solve. Paradoxically, this failure of local search to find an optimal solution is due to its myopic obsession with improvement. Most local search algorithms ensure all steps in the search process yield a better solution than is currently available. Yet, oftentimes, the local search neighborhood of the current solution does not contain any avenue for improvement and then the local search procedure is forced to stop. Iterated local search provides a workaround for this misfeature in that it transforms solutions that local search is not able to improve upon into more prolific solutions that are associated to more fertile neighborhoods. Typically, the transformation results in a solution of lower quality. Yet, most of the time, the fecundity of the new solution makes up for the loss in quality as soon as the local search is resumed.

$s_0 =$ GenerateInitialSolution
$s^* =$ LocalSearch(s_0)
repeat
  $s' =$ Perturbation(s^*, *history*)
  $s^{*\prime} =$ LocalSearch(s')
  $s^* =$ AcceptanceCriterion(s^*, $s^{*\prime}$, *history*)
until termination condition met

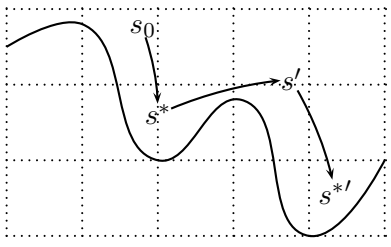


Figure 5.1: Iterated local search: Outline & illustration.

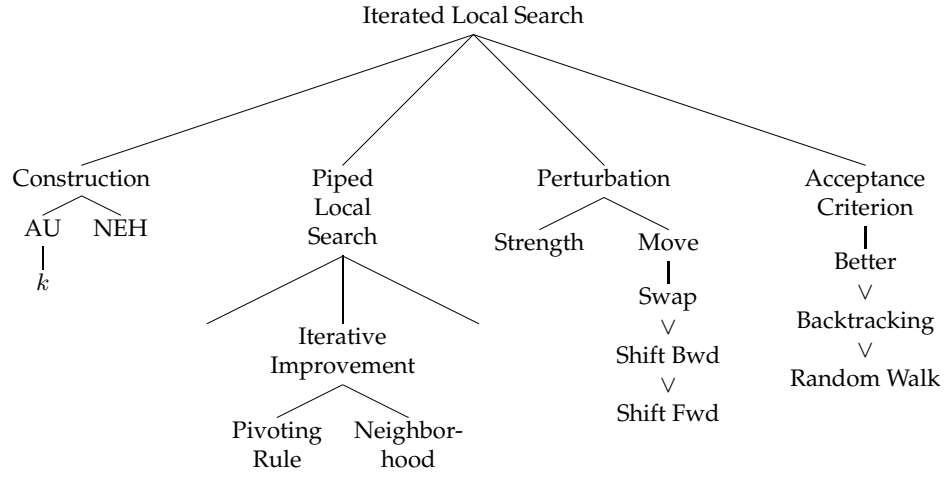


Figure 5.2: Decomposition of iterated local search employed here.

Figure 5.1 on the preceding page illustrates the workings of iterated local search. The panel on the left outlines a generic framework for iterated local search. At first, an initial solution is generated. Next, local search is applied to this solution. The solution obtained with local search is perturbed and local search is resumed. Finally, the solution thus obtained is compared with the best solution found so far and one of these solutions is selected as starting point for the next iteration. The cycle of perturbation, local search and selection is repeated until the optimal solution is found, the allocated CPU time is exceeded or some other kind of user–defined termination criterion is met. The panel on the right visualizes the first iteration of ILS. In this plot, the solid line indicates the distribution of solution qualities in the solution space. The initial solution s_0 is of relatively low quality. The next solution, s^*, is much better but also a local optimum. Perturbation yields s'. This in turn gives rise to $s^{*\prime}$. Since $s^{*\prime}$ is better than s^*, this iteration of ILS can be judged to be a success and $s^{*\prime}$ can be used as the starting point for the next iteration.

Figure 5.2 gives a comprehensive coverage of the components that constitute ILS in the context of this dissertation. Our ILS has four main components: Construction, piped local search, perturbation, and an acceptance criterion. The first two components were discussed in detail in the preceding chapters. In Chapter 3, we developed a construction algorithm that first sorts jobs according to the apparent urgency (AU) dispatching rule and then inserts each of these jobs in the solution under construction (NEH). In Chapter 4 we developed a range of local search and variable neighborhood descent algorithms out of a variety of neighborhoods and pivoting rules. The other two components, perturbation and acceptance criterion, form the extension that makes iteration of local search possible. The design and configuration of these components is the topic of Section 5.2.

It is relatively straightforward to develop a basic version of ILS. We start with

the construction heuristic from Chapter 3; local search is readily available from Chapter 4; for the perturbation, a series of random moves in one of the available local search neighborhoods can be surprisingly effective; and a reasonable first guess for the acceptance criterion is to force the cost to decrease. Although this basic version may work very well, additional tuning may yield even better results.

With respect to the initial solution, it is well known that the quality of the initial solution has a positive effect on the quality of the solution that local search manages to generate. Furthermore, a local search that starts with a good initial solution takes, on average less improvement steps and hence requires less time to execute. Therefore, the use of state–of–the art construction heuristics is commendable — especially when high quality solutions are needed quickly.

With respect to the subsequent solutions, clearly local search is the crucial component. In general, when local search A yields higher quality solutions than local search B, local search A should be included in ILS. However, when local search B is faster than local search A, it might be better to include B since it can be applied more frequently. Another aspect to consider is the ease with which the local search will be able to undo the perturbation. Less similarity between the neighborhoods for search and perturbation means less likelihood that the local search returns to the same local optimum after perturbation. So, whenever possible, local search and perturbation within ILS should be based on dissimilar neighborhoods. Better yet, for the perturbation, a neighborhood of higher order than the one used by the local search algorithm should be used.

Perturbation and acceptance criterion control the level of diversification and intensification in ILS. Perturbation often consists of a series of random moves on a solution. Often, these moves are referred to as *kicks*. Apart from the type of moves, or perturbation neighborhood, the number of moves, that is, the *strength* of the perturbation, should be specified. A strong perturbation yields a new solution that is less similar to the current solution than a weak perturbation. It is less likely that the local search will return to the same local optimum after a strong perturbation. However, most of the time, the quality of the initial solution for that local search will be lower too. Hence, the time required for the local search will be longer and the likelihood that a solution is found with better or equal quality than the current solution is also lower. Therefore, weak perturbation is preferred if the likelihood of reversal by local search can be kept at bay.

The acceptance criterion counteracts and complements the effect of perturbation. Two extreme criteria are either to accept only solutions that are *better* than the current solution or to accept *any* new solution that is generated by local search (random walk). The third way is to accept all solutions for a number of iterations before returning to the best solution that has been generated so far (backtracking). When the acceptance criterion is strict and the number of non–improving iterations is severely limited, ILS is forced to search the space close to the current solution. On the other hand, when the acceptance criterion is lenient and the number of non–improving iterations is potentially large, ILS is free to drift away considerably from the current solution. At the same time, a weak perturbation reduces the

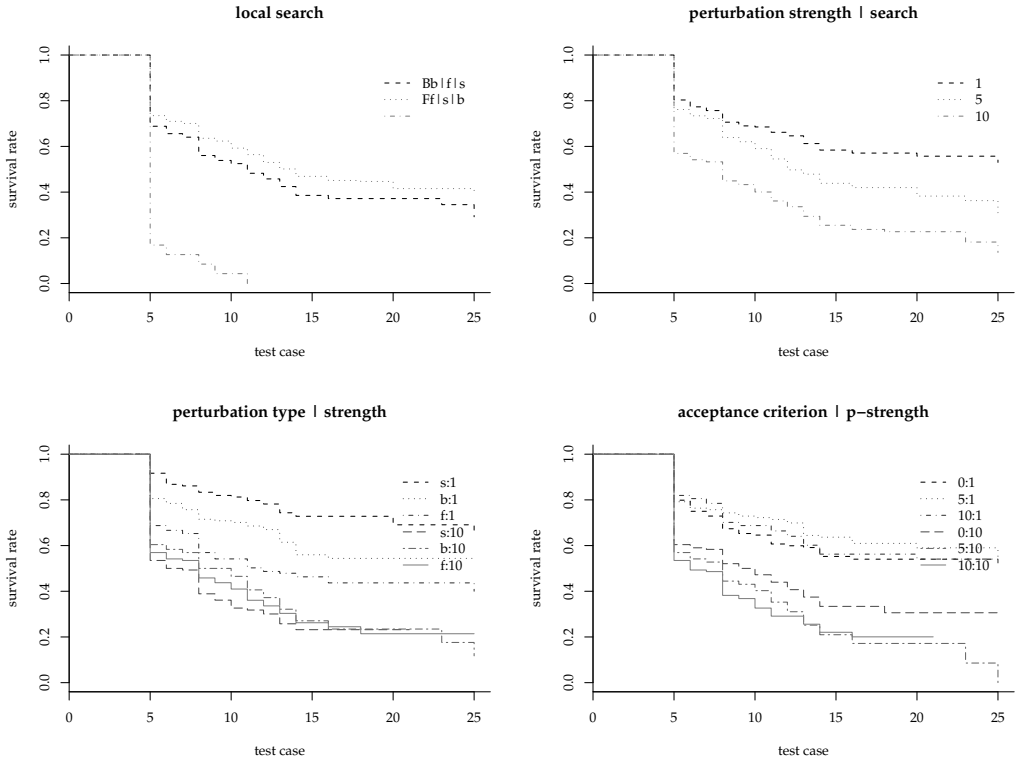


Figure 5.3: Race survival rates of iterated local search (I).

ability to drift and a strong perturbation increases the capacity for exploration.

5.2 Experimental Results

Two batches of experiments were carried out. The first has a more preliminary nature and was mainly used to find out what kind of perturbation mechanisms and acceptance criteria work well. The second was to determine which local search is best embedded in iterated local search once perturbation and acceptance criterion are fixed. The variants of iterated local search that "won" in this second batch of races are the ones that will be tested on a set of benchmarked instance in Chapter 6.

Perturbation & Acceptance

So, what kind of perturbation should be applied to the solutions found by local search? And, when should we accept a new solution?

Setup For each of the 24 instances classes, a race was done among 81 variants of ILS. The 81 configurations were specified as follows: The construction method is the one developed in Chapter 3; local search is either absent or one of $\{Bb|s|f$,

F{*f*|*b*|*s*} — these were the most frequently selected local search varieties in Chapter 4; perturbation consists of 1, 5, or 10 random moves; move type is either *b*(ackward shift), *s*(wap), or *f*(orward shift); and 0, 5, or 10 non–improving local search are accepted before ILS reverts to the best solution found so far. Each ILS candidate ran for 60 seconds on the test cases — races for single machine problems on Kioto; parallel machine races on Kika; and flow shop races on Kid.[1]

The races' assessment of ILS performance is based on the ϵ indicator which was introduced in Section 2.2 on page 35. All candidates are tested at least five times before discarding and a maximum of 100 test cases or 147×10 experiments is carried out.

Results Figure 5.3 on the preceding page contains four plots. The plots show how the survival rate of ILS candidates depends on the number of test cases and characteristics of these candidates. From Figure 5.3 the following conclusions can be drawn.

First of all, the exclusion of local search is not a good idea: In the plot in the upper left corner, survival of candidates without local search is contrasted with survival of candidates with local search. Without local search it is highly likely that the candidate is discarded within the first ten test cases and after at most 12 test cases, all variants without local search have been disarded. With local search, there is a likelihood of almost 50% of not being discarded at all.

Secondly, the perturbation strength is better kept rather limited: In the plot in the upper right corner, candidates that employ local search are grouped according to number of kicks they apply to the solution after each local search. Candidates who only apply one kick have a better chance of survival than candidates who apply ten kicks. Thirdly, the perturbation type makes a difference, and this difference is most pronounced for weak perturbations: In the plot in the lower left corner, all candidates that employ local search and apply either one or ten kicks to each intermediate solution are grouped according to the type of kick that is applied, be it swap, backward shift, or forward shift. Candidates that apply only 1 kick have a higher chance of survival when the kick is a swap kick rather than a shift kick, while forward shift is even worse than backward shift. When candidates apply 10 kicks, the type of kick that is applied matters much less.

Last but not least, the level of stringency of the acceptance criterion also makes a difference: When the perturbation is weak, the acceptance criterion should be relatively lax; when the perturbation is strong, the acceptance criterion should be strict. In the plot in the lower right corner, all candidates that employ local search and apply either 1 or 10 kicks to intermediate solutions are grouped according to the acceptance criterion which they have adopted. Among candidates that apply 10 kicks, those who only accept improving solutions perform much better than

[1] Kioto, Kika and Kid are the names of computers. Kioto has a single 1GhZ Athlon processor; Kika has two 2GhZ Athlon processors; and Kid has two 930MhZ Pentium III processors. All three are running Linux 2.24 from the Debian distribution.

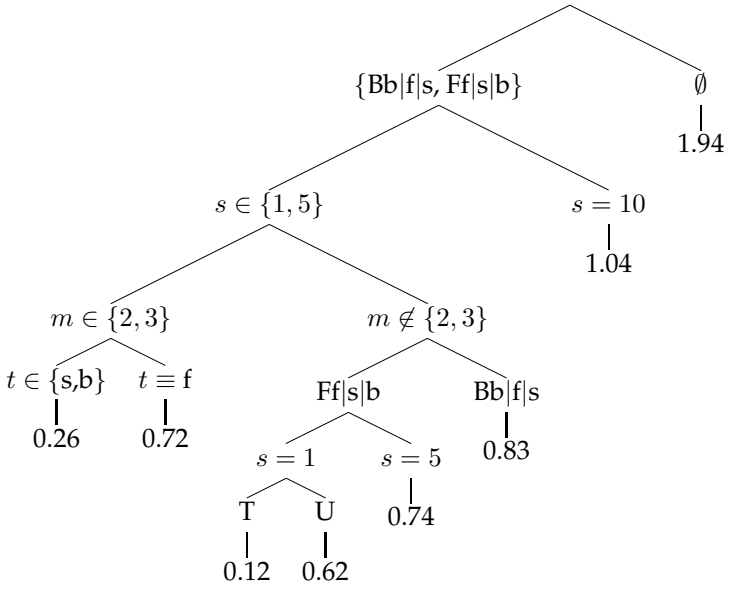


Figure 5.4: Factors in race survival: s is the perturbation strength; m is the number of machines in the problem; t is the perturbation type; and T and U are shapes of the objective function.

those who allow themselves to wander off. Among candidates that apply 1 kick, the differences are less pronounced. The effect that is visible, however, is that candidates with a moderate acceptance criterion, backtracking after 5 non-improving solutions, stand a better chance of survival than those candidates who opt for a more extreme criterion.

Additional information on the races is given in Appendix A.3.1. Table A.4 in that appendix indicates for each race and for each of the components of iterated local search how often they can be found in the iterated local search algorithms that survived the racing experiment. The table corroborates the findings from Figure 5.3 in that the permutation strength should not be too strong and the acceptance criterion should not be too permissive.

Figure 5.4 is a decision tree constructed on the basis of the results from the races. For all candidates involved in the races, it predicts their chance of survival as the basis of their configuration. The numbers at the leaves correspond to the hazard that candidates run to be discarded at each step of the race. A high number corresponds to a low probability of making it to the end of the race. The nodes in between branches indicate what feature of the candidate contributes to its success or failure. The first split in the tree is between candidates with and candidates without local search. Hence, local search must be the most important contributor to success. Candidates with local search are subsequently split between those who apply few kicks to intermediate solutions (good) and those who don't (bad). The other splits go to show that it matters what kind of kick is applied when the test

cases are instances from environments with two or three machines in parallel and that it matters what kind of local search is applied when the test cases are instances from environments with a single machine, four parallel machines, or a flow shop. All in all, Figure 5.4 confirms the observations from Figure 5.3 and so we can conclude that iterated local search is indeed better off with a powerful local search component and a perturbation that is neither too weak nor too strong. The fact that the acceptance criterion does not show up in Figure 5.4 indicates that all three variants of the criterion are acceptable in that they affect the performance of ILS less than perturbation or local search.

Local Search

Now that we have an idea of what kind of perturbation and acceptance criterion to pick, let's pay some more attention to the local search component.

Setup Local search is the most important aspect of ILS. Therefore, choosing the right variant of local search is crucial. In contrast, ILS is relatively robust with respect to acceptance criterion and perturbation. The next batch of experiments consists of twenty–four races in which 147 candidates are compared against each other. The candidates differ in terms of local search only. That is, for each of the local search candidates that was selected in one of the races in Chapter 4, there is an ILS candidate. Just like in the previous races, the initial solution is constructed according the procedure specified in Chapter 3, but unlike in the previous races, this time around each candidate employs the same perturbation and acceptance criterion. More specifically, the acceptance criterion is strict in that only better solutions are accepted; the perturbation is adaptive and stochastic: ILS starts with a perturbation strength of 1, but whenever the local search fails to yield an improvement, the perturbation strength is incremented with 1. The strength is reset to 1 when a better solution is found and the perturbation strength is not allowed to exceed 10. The type of perturbation is either swap, backward shift, or forward shift, where each of swap, backward shift and forward shift has an equal probability of being picked.

Like in the previous races, the races' assessment of ILS performance is based on the ϵ indicator. All candidates are tested at least five times before discarding and a maximum of 100 test cases or 147×20 experiments is carried out.

Results The iterated local search candidates that were not discarded during the race are listed in Appendix A.3.1. It is interesting to compare these results with the results from the races among local search candidates in Appendix A.2. On the one hand, the iterated local search races are more inclusive in that a larger number of candidates tend to be selected. On the other hand, the iterated local search races are more restrictive in that candidates with a local search component of exactly three components with each of the three neighborhood moves represented at least once are remarkably predominant. On the one hand, the order of the neighborhoods and occasional redundancy seem to matter less for iterated local search. On

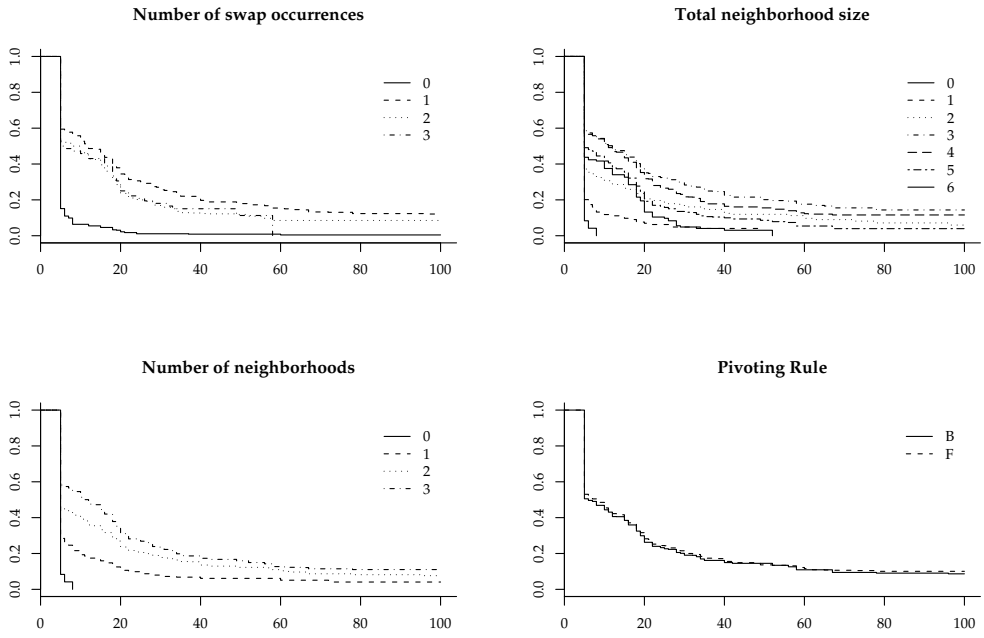


Figure 5.5: Race survival rates of iterated local search (II).

the other hand, local search variants that do extremely good in terms of run time or solution quality alone survive the local search races but do not survive iterated local search competition.

Figure 5.5 shows a number of plots that depict the relationship between survival and local search specification. The plot in the upper left corner depicts survival fits for local search candidates with various frequencies of swap moves. When there is no neighborhood in the local search that contains a swap move, the solid line in the plot, then the chance of survival for the iterated local search candidates is rather dim. When the swap moves are scanned in three distinct neighborhoods, the dashed–dotted–line in the plot, then the iterated local search candidate will eventually drop out as well. The highest chance of survival have those candidates that scan the swap moves exactly once (dashed line). Hence, swap should be included, but redundancy is penalized.

The plot in the upper right corner depicts survival fits for local search candidates according to their combined neighborhood size. That is, all distinct moves in the neighborhoods are counted and summed. Thus, a local search such as *Ffsb||* has size 3 and *Bb|fs|* too, while a local search component like *Ffs* or *Fs|b* has size 2. In this plot, the dashed-dotted line that corresponds to size 3 dominates all others. Apparently, all three move types should be included and, again, redundancy is penalized.

The plot in the lower left corner depicts survival fits for local search candidates

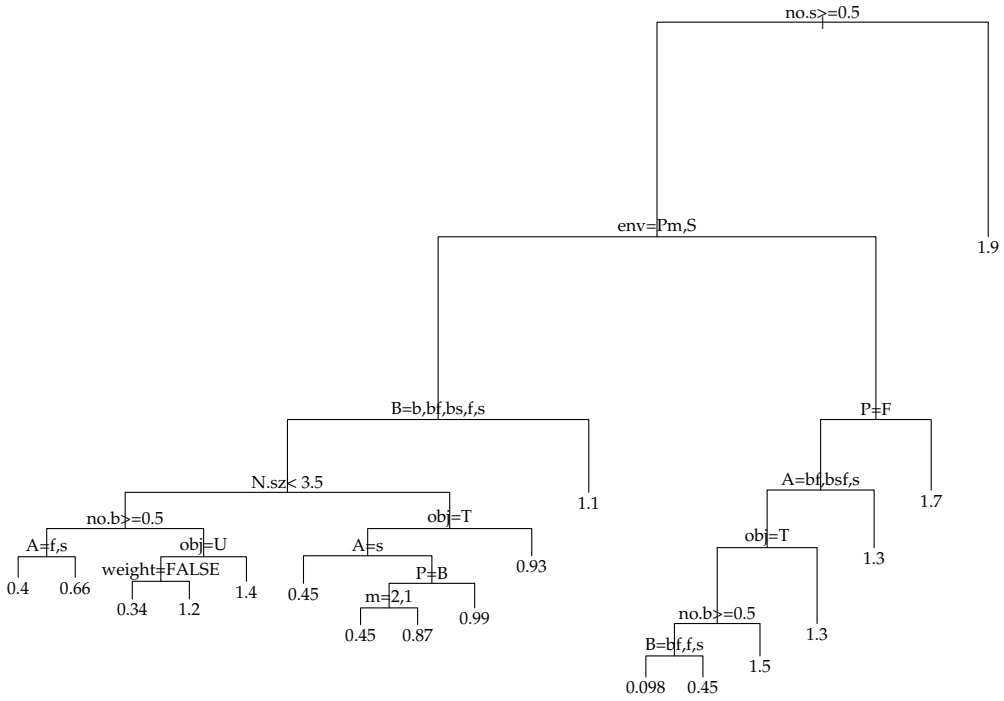
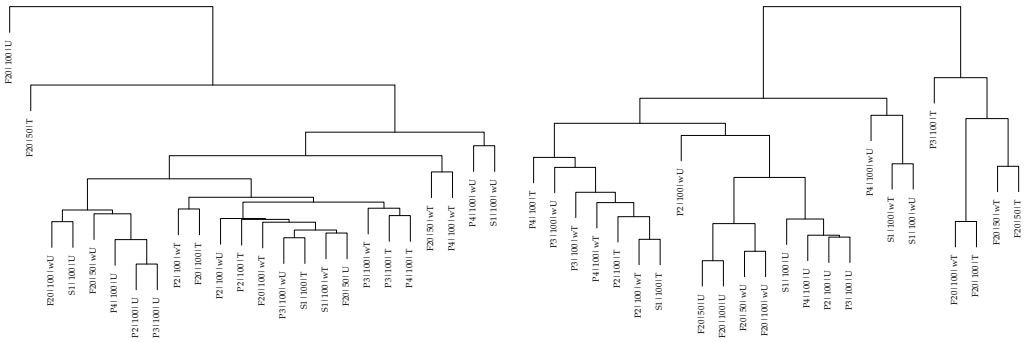


Figure 5.6: Decision tree of race survival with respect to local search features. Iterated local search candidates are characterized by their local search component, where *no.x* is the number of times that the *x* move occurs in the local search, *N.sz* is the combined neighborhood size, *A*, *B*, and *C* are the first, second, and third neighborhood and $P \in \{F, B\}$ denotes the choice of pivoting rule. Furthermore, *env*, *obj*, *weight*, and *m* denote problem environment, objective function, weight and number of machines of the test cases.

according to the number of neighborhoods that they are composed of. From this, we can conclude that iterated local search that uses local search consisting of various neighborhoods outperforms iterated local search with single neighborhood local search: Beware of the monolith neighborhood.

Finally, the plot in the lower right corner illustrates that the choice of the pivoting rule does not matter in the context of iterated local search.

Figure 5.6 shows the decision tree that has been generated on the basis of the experiments. The tree is like the tree of Figure 5.4 except that this time the tree focuses on features that describe the local search component rather than perturbation or acceptance criterion. Apparently, considering swap moves in local search is critically important to survive the races. Beyond that, the flow shop environment exerts other pressures on iterated local search candidates than the parallel or single machine environments. In the flow shop environment, the choice of the pivoting rule matters. And if the objective is to minimize total tardiness or total



(a) Perturbation & Acceptance Races (b) Local Search Races

Figure 5.7: Clusters of races with similar candidate eviction patterns.

weighted tardiness, it is also important to include the backward shift among the moves to be considered by local search. In contrast, in the parallel and single machine environments, it may actually be beneficial to exclude the backward shift if the objective is to minimize the total number of tardy jobs.

5.3 Patterns & Deviation

In the previous section we already found out which configurations of iterated local search work best on our scheduling problems and why and so this section is bound to be a bit of an afterthought. There are three questions the still merit addressing. Firstly, it would be interesting to see whether the hierarchy of problems in Figure 1.3 on page 14 is in any way reflected in the evolution of the races. Secondly, it would be interesting to see how similar or dissimilar the candidates in both batches of races are. Thirdly, it would be interesting to see how robust the conclusions drawn in the previous section are with respect to choice of selection procedure.

Problem Clusters

Figure 5.7 contains two hierarchical clusters, one for each batch of races performed in the previous section. The clusters were obtained by running the hclust procedure on a distance matrix built from a matrix that contains for each race and candidate the number of test cases the candidate was tested against before being discarded. From Figure 5.7 we can deduce that the second batch of races is more problem dependent than the first batch in that the selection of candidates varies more dramatically from one race to another as the relatively more “neat” division in the cluster in the second figure testifies. From Figure 5.7 we can also deduce that problem size does not matter. For the 50 job flow shop races and the 100 job flow shop races invariably appear close to each other. Finally, we can deduce the relative

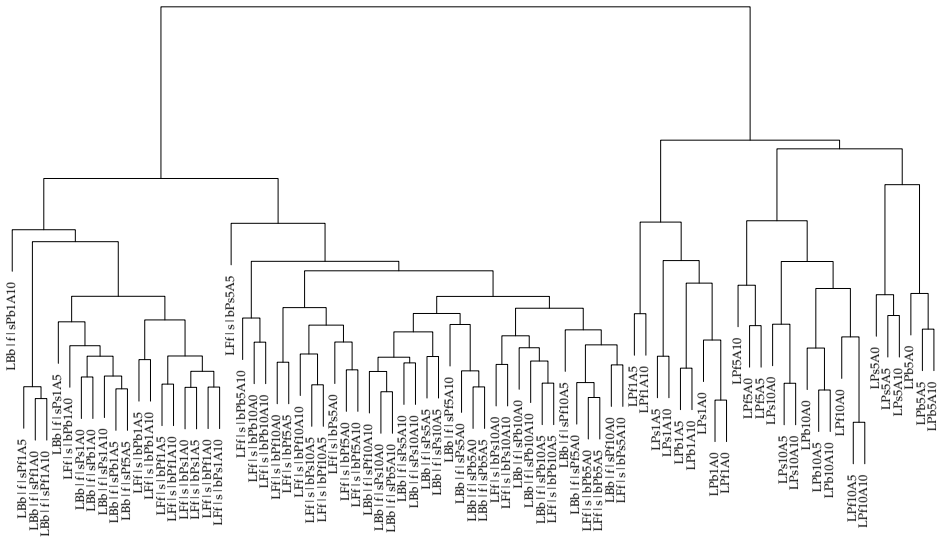


Figure 5.8: Clusters of candidates with similar race success (I)

importance of problem characteristics. Unfortunately, both clusters do not send an unequivocal message. In the first batch of races depicted on the left, problems with the unit penalty function are mostly neatly separated from problems with the total tardiness penalty function, but beyond that problems seem to be happy to mingle. In contrast, in the second batch of races depicted on the right, the shape of the penalty function seems again to be the most important distinguishing factor. Beyond that, the distinction between flow shop environment and non flow shop environment matters while the distinction between single machine and parallel machine environments is only gradual. At the same time, the addition of weights seems to have a greater impact in the simple single machine environments than in the more complex flow shop environments. Figure 4.7 on page 75 gives yet another slightly different picture for races between candidates of piped iterative improvement. Yet, as far as the hypercube in Figure 1.3 might have conveyed the impression that each dimension along which the problems are differentiated are equally important, the cluster in Figure 5.7 clearly shows that such a thing is not true if the behavior on races is taken as reference.

Candidate Clusters

Figure 5.8 and Figure 5.9 on the next page show hierarchical clusters of the iterated local search candidates in the two batches of races. The cluster are based on the ϵ scores of the candidates on the first 10 test cases. Figures 5.8 and 5.9 are strikingly different in shape: In Figure 5.8 three main clusters can easily be distinguished. In contrast, Figure 5.9 consists of one big cluster and a few outliers. Figure 5.8 reflects the findings of Section 5.2 as the main split is based on local search and subsequent splits are based on perturbation strength, and, to a lesser extend, perturbation

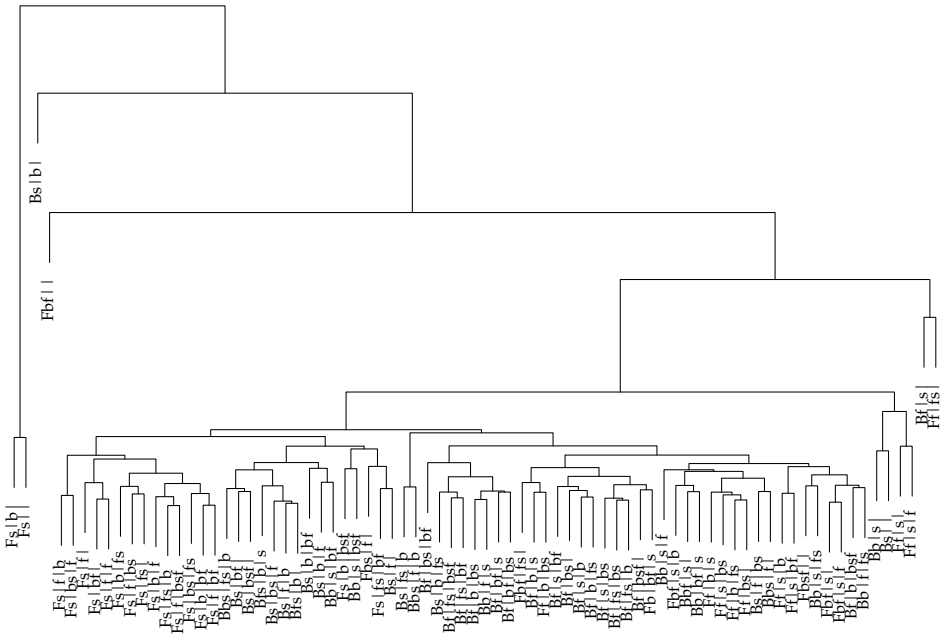


Figure 5.9: Clusters of candidates with similar race success (II)

time — just like in the decision tree of Figure 5.4. In the absence of local search (the rightmost cluster), perturbation and acceptance act as a substitute local search and the sub–clusters show that some combinations of perturbation and acceptance criterion have more success in this task than others.

The relative lack of structure in Figure 5.9 may be due to the fact that the outcomes of the batch of races the cluster is based upon is more problem dependent (in Figure 5.6, the problem environment is a branching point quite close to the root). Also the fact that there are 6-10 candidates that perform really badly in comparison with the rest, the outliers in Figure 5.9, may skew the overall picture. Yet, even in the main cluster the order of the candidates seems to be rather arbitrary (although the initial neighborhood in the local search component seems to have some effect). So, perhaps there is simply not that much difference between most of the tested candidates and each of them, except for the outliers, could be expected to perform reasonably well.

Robustness of Results

The selection in the races is based on performance of the candidate measured by the ϵ–indicator. The ϵ–indicator is a relatively intricate way to measure performance. Therefore, it is valid to question whether a simpler measure, say one based on solution quality after a fixed run time, would produce hugely different results.

Table 5.1 on the facing page may go some way in providing the answer. Table 5.1 indicates how ranks obtained when performance is measured with the ϵ indicator

Table 5.1: Correlation between several rankings of ILS candidates: Ranking according to ϵ–indicator (ϵ) and ranking according to solution quality after 1, 2, 4, 8, 16, and 64 seconds ($sq(i)$).

ϵ	0.47	0.56	0.61	0.64	0.68	0.66	0.61
	$sq(1)$	0.80	0.71	0.64	0.58	0.56	0.55
		$sq(2)$	0.86	0.77	0.69	0.67	0.65
			$sq(4)$	0.88	0.78	0.74	0.72
				$sq(8)$	0.87	0.82	0.80
					$sq(16)$	0.93	0.90
						$sq(32)$	0.96
							$sq(64)$

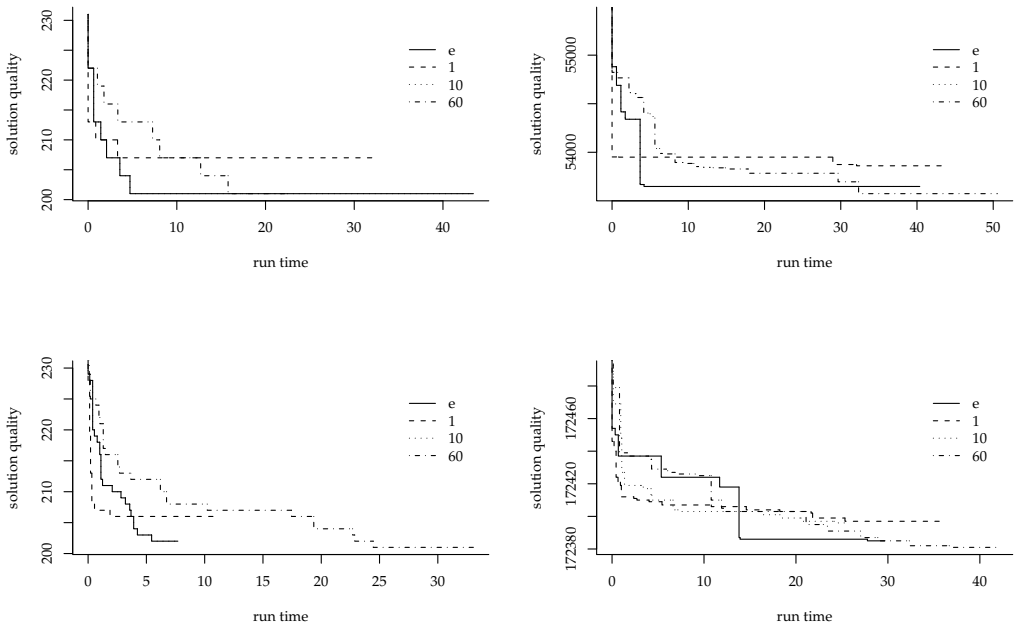


Figure 5.10: Comparison of traces of iterated local search variants with best ϵ value (e) or best solution quality after 1, 10, or 60 seconds.

correlate to ranks based on the quality of the best solution that has been found within a certain amount of time. The correlations are based on results of the first five test cases of all 24 races in the first batch of races. What can be deduced from the table is that the ϵ–indicator is most correlated with the scores on run times that are no too small and no too large. Furthermore, the correlation between subsequent measurements of solution quality is quite high — as one would expect — and the high correlations between measurements after long run times indicate that the ranking is settled after a while.

Still, Table 5.1 also shows that the selection of candidates depends on run time and that the ϵ–indicator seem to strike a magical balance between run time and solution quality. If the outcomes of the races are different for different performance measurement methods, the whole analysis might be different. Fortunately, when I tried to cluster candidates on the basis of solution quality rankings after fixed run times, structures very similar to the ones based on the ϵ–indicator appeared.

While, Table 5.1 gives the general impression that the ϵ–indicator strikes a fine balance, Figure 5.10 plots the traces for ILS variants for specific instances. Each plot in the figure corresponds to a randomly picked instance and the lines depict the traces of run times and solution qualities produced by the ILS variant that performed best on the instances according to the ϵ–indicator, or based on the solution quality found after 1, 10, or 60 seconds. The first plot in the upper left corner is comfortably reassuring in that the trace of the variant picked by ϵ is better than or equal to the other traces all of the time with the exception of the first few seconds where the trace of the variant that had the best performance after one second dominates the others.

Unfortunately, the other three plots look less reassuring. Overall, the traces picked by the ϵ indicator still performs best. But, each time, the variant that is picked for its performance after 60 seconds manages to undercut the ϵ variant at the last moment. Thus, there is a distinct possibility that the variant selected by ϵ is not the best choice is run time is not an important consideration with respect to the goal of attaining the best possible solution quality.

5.4 Summary

After the investigations into the development of construction heuristics and local search in Chapter 3 and 4, this chapter set itself the task to find out how best to configure iterated local search for tardiness related scheduling problems. For this purpose, several racing experiments between promising candidates were carried out.

Iterated local search consists of three components: local search, a perturbation mechanism, and an acceptance criterion. A first set of races focused on the latter two components and from subsequent analysis it was deduced that the acceptance criterion should be rather stringent whilst the perturbations should be rather weak. The next set of races pitted candidates with different local search compo-

nents against each other. Here, it was found that, in the context of iterated local search, choice of pivoting rule does not matter much and that it is generally a good thing to try and include all three types of neighborhood moves in the local search. Specifically the inclusion of the swap move proved to be important. Furthermore, it was found that searching through multiple small neighborhoods tends to yield better results than search through a small number of big neighborhoods.

The last section of the chapter addressed a couple of questions beyond the immediate concern for getting at the best configuration for iterated local search. It turns out that way that the races evolve depends on the type of instances on which the iterated local search candidates are tested. What's more, the similarities between the races reflect a problem hierarchy where the specification of objective function has a greater impact than the specification of the machine environment and where the disruptive effect of the weights seems to become less pronounced as the problems grow more complex. Meanwhile, cluster analysis confirmed that the overall findings derived from the races were rather robust with respect to the specification of criteria for performance measurement and the choice of tools for analysis. However, the choice of method of performance measurement does matter with respect to the final outcome of the races as the rankings used in the races correspond most to rankings obtained from the quality of solutions found by the candidates within a reasonably short, yet still sufficient, run time.

Part III

Results

6 Benchmark Performance

In this penultimate chapter, we take a closer look at the performance of the best variants of iterated local search that came out of the investigations described in the previous chapters. In order to provide future researchers with an easy reference, a benchmark set is defined and the performance of iterated local search on this benchmark set is delineated. In so far as possible, a comparison is drawn between iterated local search's performance and the state of the art. Finally, a summary of the main findings is provided at the end of the chapter.

6.1 Experimental Setup

Performance assessment involves candidates, test cases, and a computer. This section describes which candidates, test cases, and computer were used for the final tests.

Algorithm Selection

The last batch of races in the Chapter 5 was some sort of grand finale in the development and selection of iterated local search for the scheduling problems with tardiness penalties which we are interested in in this dissertation. The variants of iterated local search that survived these races are listed in Appendix A.3.2. For each problem class, a list of variants is given ordered in accordance with the overall rank that they were awarded in the race. Because the iterated local search variants only differed with respect to the local search component that was employed, only a local search component identifier is used to describe the variants. Table 6.1 on the next page lists for each race of this last batch, and hence for each problem class, which iterated local search variant "won" the race. In the context of this chapter, Table 6.1 describes which local search component is employed in the variant of iterated local search that is tested on the benchmark set.

Apart from local search, iterated local search features an acceptance criterion, a perturbation mechanism, and a procedure to construct the initial solution. The acceptance criterion is set such that only solutions that are strictly better than previously found solutions are accepted as new starting points. The perturbation consists of a series of moves that are applied one to ten times depending on how many previous local search applications in a row failed to produce an improvement. The move used for perturbation is either a swap move, or a backward shift move, or a forward shift move as one of each is randomly selected each time a random move

Table 6.1: Selection of local search component in iterated local search depending on problem class.

	1	$P2$	$P3$	$P4$	F_{20}^{50}	F_{20}^{100}
$\sum U_j$	Ff|b|s	Bb|s|bsf	Fs|f|bs	Ff|bf|s	Ff|s	Ff|s
$\sum T_j$	Bf|bf|bs	Bbf|s	Fs|f|b	Fb|bs|f	Fbf|s	Fbf|s|b
$\sum w_j U_j$	Bf|s|b	Bb|f|s	Bbf|s	Bf|bs	Ff|s	Ff|s
$\sum w_j T_j$	Bf|s|b	Bb|s|f	Bs|b|bsf	Bs|f|bsf	Fbf	Fs|b|f

is applied. Finally, the initial solution was generated with the construction procedure of Chapter 3. This procedure orders jobs according to the apparent urgency dispatching rule and then uses this order to subsequently insert jobs in the solution sequence under construction. A decision tree, more in particular the one pictured in Figure A.3 on page 123, is used to determine an adequate value of apparent urgency's look–ahead parameter for each problem and instance class.

Benchmark Set

Unfortunately, no benchmark set is readily available for each of our scheduling problems and so we are forced to establish a new benchmark set for most problems. Luckily, a benchmark set is available for $1||\sum_{j=0}^{100} w_j T_j$ [Beas 90] and for $Fm|\text{prmu}|C_{\max}$ [Tail 93]. From these sets, the benchmark instances for all other problems can be generated relatively easily.

The $1||\sum_{j=0}^{100} w_j T_j$ benchmark set consists of 125 instances with 100 jobs each. The set, which can be found at ORLIB [Beas 03], was generated as follows: For each job j, a processing time p_j was randomly drawn according to a uniform distribution on the interval $[1, 100]$, due dates were randomly drawn according to a uniform distribution on the interval $[(1 - \text{TF} - \frac{\text{RDD}}{2}) \cdot \sum p_i, (1 - \text{TF} + \frac{\text{RDD}}{2}) \cdot \sum p_i]$, where TF (tardiness factor) and RDD (range of due dates) are parameters.[1] There are five instances for each pair of TF and RDD from the set $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. Finally, the weights, randomly drawn according to a uniform distribution of integers between 1 and 10, were added.

The $1||\sum_{j=0}^{100} w_j T_j$ benchmark set can be used straightaway for $1||\sum_{j=0}^{100} w_j U_j$. In case of $1||\sum_{j=0}^{100} T_j$ and $1||\sum_{j=0}^{100} U_j$, the weights are set to one.

The set can also be amended for parallel machine environments. The only adjustment that is needed for these environments is a division of the due dates by the number of machines in the environment. The reason for this adjustment is that the span between the entry of the first job into the system and the exit of the last job out of the system, the makespan, can be expected to decrease by the same factor and since the tardiness factor are defined as the deviation of the average due date from the makespan, the due dates have to get shorter as well.

[1] Note that, strictly speaking, due dates may be negative if $\text{TF} + \text{RDD}/2 > 1$. However, in the $1||\sum_{j=0}^{100} w_j T_j$ benchmark set, negative due dates are set to zero. Note that this was not done for the test instances that were generated for algorithm development and testing purposes in previous chapters.

For the flow shop problems, a slightly more involved procedure is used to generate the benchmark sets. The processing times are taken from instances ta50-54 and ta80-84 of the $Fm|\text{prmu}|C_{\max}$ benchmark set [Tail 93]. These instances contain processing times for 50 jobs on 20 machines and 100 jobs on 20 machines respectively. To the processing times, due dates are added. More specifically, for each 'ta' instance, 25 due date distributions are generated in the same fashion as in the $1||\sum_{j=0}^{100} w_j T_j$ benchmark set except that this time around the interval from which the distribution is drawn is defined with respect to the best known makespan of the 'ta' instance rather than the sum of processing times.[2] Finally, weights are added to the 125 50–job and 125 100–job instances thus obtained. As it happens, these weights are the same as the weights in the 50–job and 100–job $1||\sum w_j T_j$ instances.

Note that none of the instances of the benchmark set described here have been employed as test cases in any of the races of the previous chapters. There, instances randomly generated along the lines set out in Section 1.3 were employed.

Computer Environment

All tests were carried out on a computer with two Intel 2.4GHz processors running Debian Linux kernel 2.4.24-xfs. Iterated local search is run ten times on each instance and is allowed 5 minutes per run.

6.2 Performance Characterization

The median deviation from the best solution is reported for iterated local search after 1, 60, and 300 seconds for all instances individually in Appendix B. This section is concerned with the aggregate results.

Run Time Distributions for Instances 86–89 in $P3||\sum T_j$

Run time distributions have proved to be a valuable way to characterize the performance of iterated local search on individual problem instances [Stut 98]. To obtain a run time distribution, rerun your algorithm on the same problem instance a number of times and then tabulate for each run the first time at which the algorithm reached a pre-defined target. Typically, the target is to find a solution with a quality equal to the quality of the best known solution, but the target can also be to reach a quality within a certain deviation of the best known or anything else that your algorithm's performance can be compared against. Figure 6.1 on the next page depicts the run time distributions for iterated local search on four instances of the $P3||\sum T_j$ scheduling problem. Here, run times on the x-axis are plotted against success rates on the y-axis for three targets: 0% deviation from the best known, 1% deviation from the best known, and 10% deviation from the best know. As one would expect, the likelihood that iterated local search comes within 10% of the best known is consistently higher than the likelihood that the algorithm

[2] Plus, this time around negative due dates remain unchanged and are not set to zero.

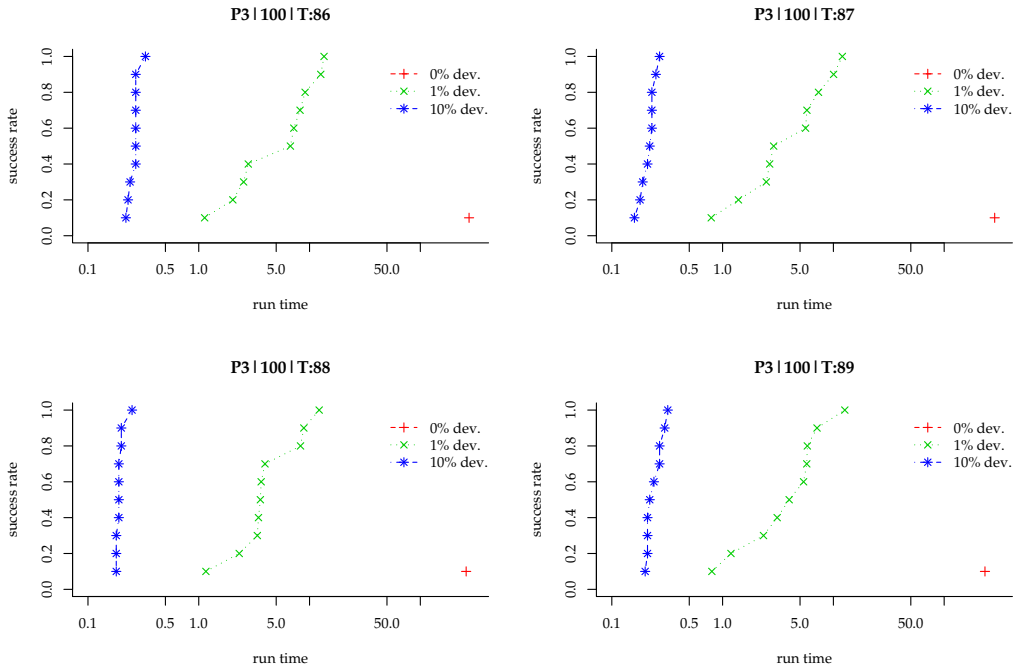


Figure 6.1: Run time distributions for instances 86–89 in $P3||\sum T_j$.

actually finds the best known solution. Furthermore, note that for these particular instances it is quite unlikely that the best known solution quality is also the best possible solution quality because the best known value is reached only once and at around the cutoff time of 300 seconds in each of the plots.

The run time distributions for the four distinct instances are relatively similar to each other in that in all cases a solution is found within 10% from the best known within 0.5 seconds. Moreover, in all cases it takes between 0.5 and 50 seconds to come within 1% of the best known values. And, in all cases the best known solution is found at the end and only once. Thus, shapes and ranges are similar and that is not entirely unexpected since all four instances were generated with the same tardiness factor and range of due dates. In fact, similar similarities emerge among instances in classes characterized by other problem specifications and tardiness factor and range of due dates combinations.

The similarity among run time distributions is important as it allows us to lump together the results for all instances generated on the basis of the same range of due dates and tardiness factor. And so, rather than considering all 3000 instances individually, we can confine ourselves to a mere 600 instance classes.

Proportion of Runs on Target per Instance Class

Figure 6.2 on page 104 gives an indication of the performance of iterated local search for each of these 600 instance classes. The figure consists of four level plots.

Each level plot contains 600 boxes and the tone of gray of each box indicates how successful iterated local search was in attaining the target on the corresponding instance class. Black stands for no success and white stands for complete success.

The upper left plot indicates how often iterated local search manages to find the best solution within 1 second. This plot is very dark because one second is not an awful lot of time. However, in defiance of the general picture, iterated local search does manage to solve the instances with a small tardiness factor in time in all environments except for flow shop environment as is testified by the white bars in the plot.

Next, the upper right plot indicates how often iterated local search manages to find the best solution within one minute. Compared to the plot on the left, this plot is very bright and so we can deduce that iterated local search manages to find the best solution within one minute for many instances. Since the best known solution for all problems except for $1||\sum U_j$, $1||\sum T_j$, and $1||\sum w_j T_j$ is the solution that was reached after five minutes, a light shade also indicates that iterated local search does not improve its solutions much in the remaining four minutes and a dark shade indicates that the best known solution is typically found after more than one minute. Iterated local search seems particularly well adapted to $1||\sum w_j T_j$: the square for that instance class is almost devoid of ink.

Below, the left and right plots indicate how often iterated local search manages to find a solution within one percent deviation from the best known after one second for the plot on the left and after one minute for the plot on the right. Clearly, coming within one percent reach of the best is much easier than actually finding the top quality solution itself.

Impact of Problem Specification on Likelihood of Reaching Target

Now that we have an impression of the big picture, the next step is to build a model of iterated local search performance. The decision trees in Figure 6.3 on page 105 provide models of this kind. The decision trees are based on a survival analysis interpretation of the data. In this interpretation, the iterated local search patient dies when it reaches its target. Thus, the decision trees predict the cumulative hazard of reaching the target — obtaining a solution of the best known quality for the tree above, within one percent deviation of this quality below — and a high number at the leaf indicates a high likelihood of success for iterated local search on instances corresponding to the leaf at any particular point in time.

From the decision trees induced from the experimental data we can deduce that there is a big difference between flow shop and non flow shop problems. For, in both trees $m > 12$ splits flow shop problems from other problems right at the root. When deviation from the best, rather than the best itself is considered as target, the shape of the objective function also plays a role as is evidenced by the appearance of obj $= U$ early in the second tree. A possible explanation for this is that with a unit penalty a misassignment of one job already yields a 1% deviation from the optimum while potential reductions in the tardiness of this job could well be much less than 1% of the total tardiness of all jobs. Apart from that, the tardiness factor

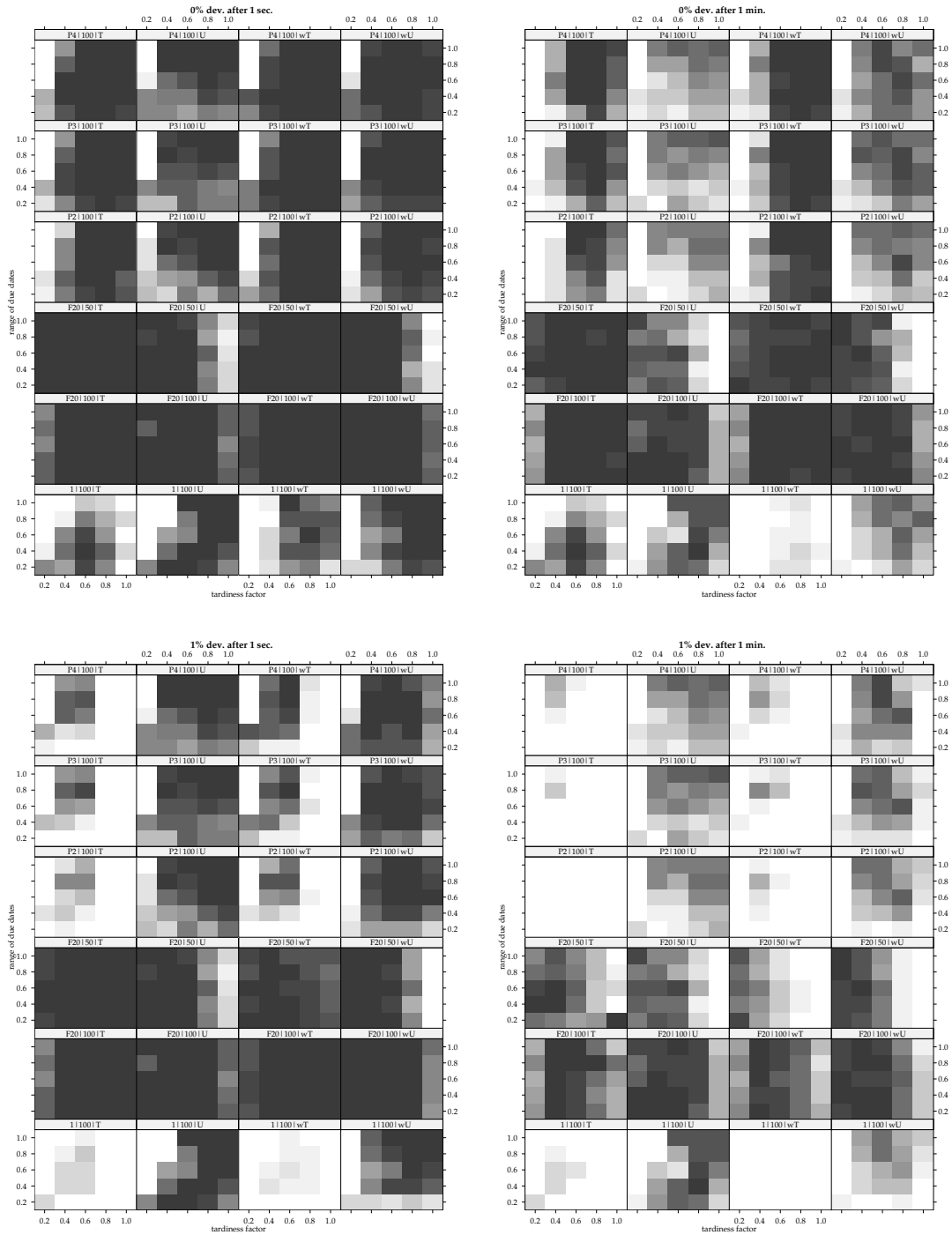


Figure 6.2: Proportion of runs on target for each instance class. Shades of gray correspond to the success rates with black for no success and white for complete success. The targets are finding the best known solution within 30 seconds (left upper corner), within 300 seconds (upper right corner), within 1% from best known within 30 seconds (lower left corner), and within 1% within 300 seconds (lower right corner).

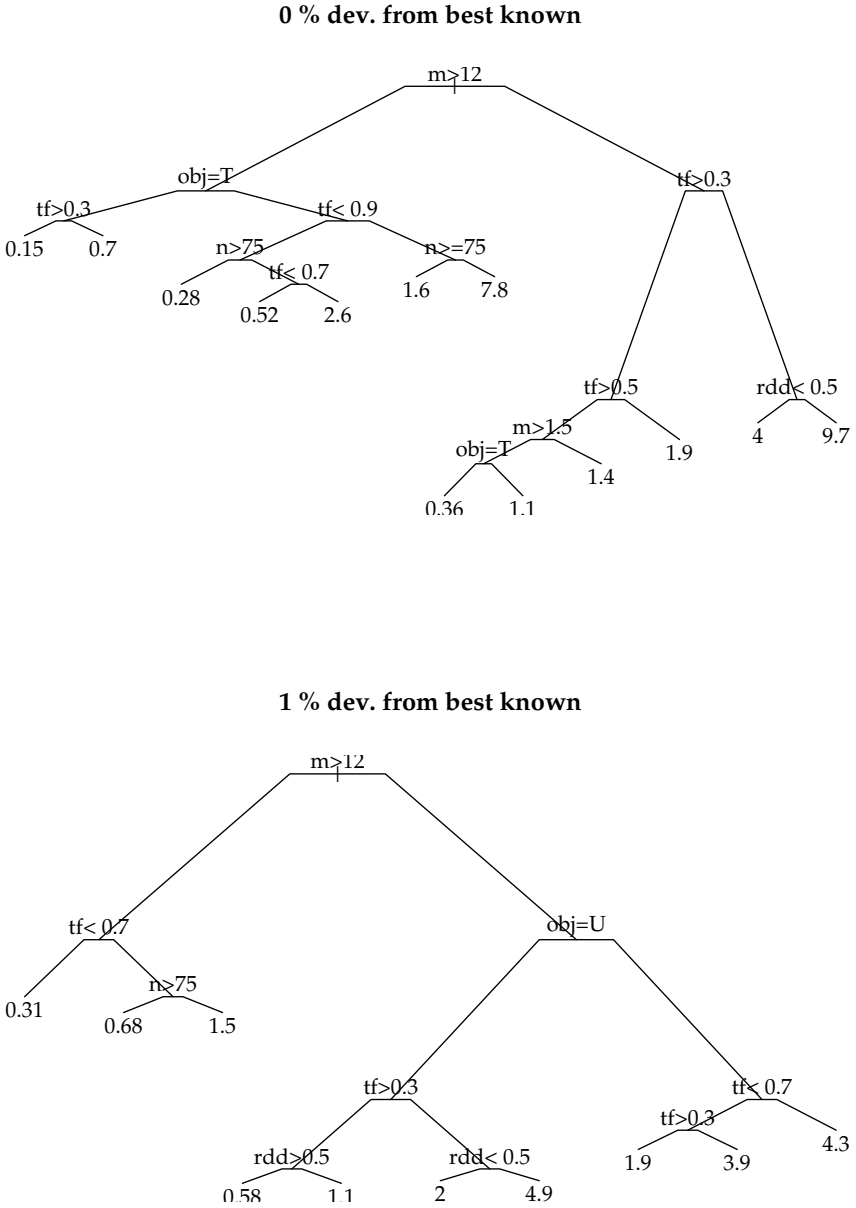


Figure 6.3: Impact of problem specification on likelihood of reaching target. Target is specified above the decision trees. Splitting criteria: number of machines (m); number of jobs (n); objective function (obj); tardiness factor (tf); range of due dates (rdd);

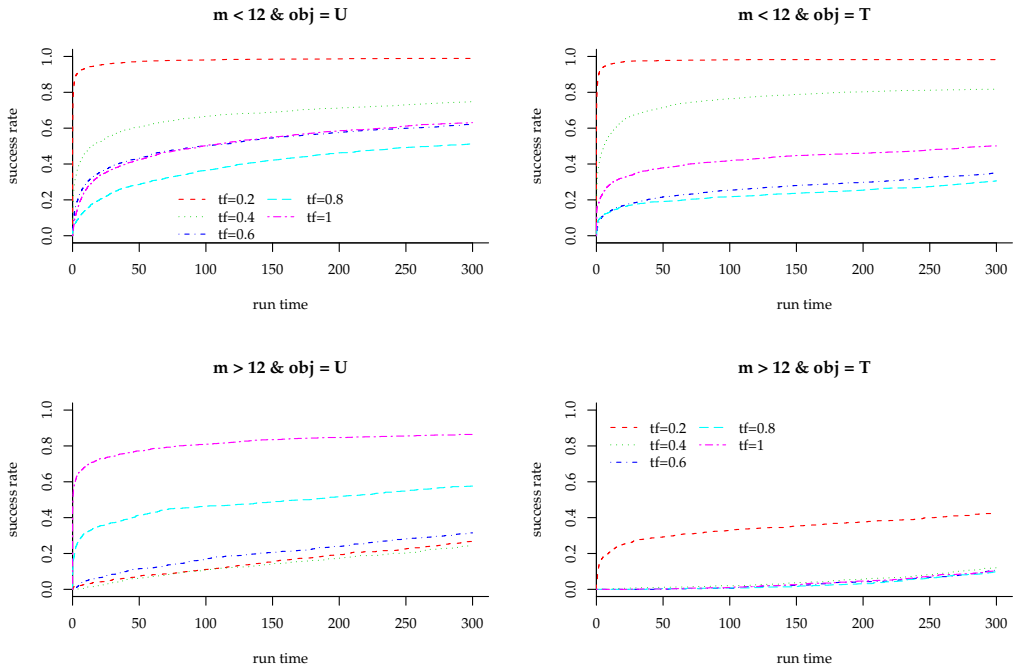


Figure 6.4: Impact of tardiness factor on instance difficulty.

(tf) seems to be crucial. Flow shop problems with the total (weighted) tardiness objective functions are most difficult for $tf > 0.3$. Flow shop problems with unit penalty objectives are easiest with $tf > 0.9$. Meanwhile, non flow shop problems are very easy when $tf < 0.3$.

Impact of Tardiness Factor on Instance Difficulty

Figure 6.4 depicts the effect of tardiness factor more clearly. It shows the plots of survival curves for four groups of problems split according to tardiness factor. The plot in the upper left corner indicates how tardiness factor impacts the success of iterated local search on unit penalty problems in the single or parallel machine environments. Iterated local search almost always solves instances with a tardiness factor of 0.2. In contrast, if the tardiness factor is 0.8, more than fifty percent of the instances remain unsolved, even after 300 seconds. The difficulty of instances with a tardiness factor of 1.0 and those with a tardiness factor 0.6 is about the same for iterated local search, whilst instances with a tardiness factor of 0.4 are slightly easier.

Next, the plot in the upper right corner indicates how tardiness factor impacts the success of iterated local search on total (weighted) tardiness problems in the single or parallel machine environments. The order of the survival curves corresponding to the tardiness factors is the same as in the plots to the left. But, at the same time, the tardiness objective makes the difference between the tardiness fac-

tors more pronounced: Instances with a tardiness factor of 1.0 are no longer equal to instances with a tardiness factor of 0.4 and the curves of tardiness factor 0.4 and 0.8 are further apart in this plot than in the plot to its left. Furthermore, except for instances with a tardiness factor small or equal to 0.4, the success rate of iterated local search is worse on instances with a total tardiness objective than on instances where a unit penalty is applied.

Below, the plots show the effect of tardiness factor on instances difficulty for the flow shop environment. The plot in the lower left corner gives the survival curves for flow shop instances with a (weighted) unit penalty objective function and the plot in the lower right corner gives the survival curves for flow shop instances with a total (weighted) tardiness objective function. Overall, the curves in these plots remain lower, reflecting the fact that flow shop instances are harder to solve than non flow shop instances. In the plot in the lower right corner, the curves for all tardiness factors apart from 0.2 reach a success rate of 0.1 after exactly 300 seconds. Since 300 seconds is the point in time at which all runs were stopped, and a success rate of 0.1 corresponds to exactly one run on target, the plot tells us that for the instances where the tardiness factor exceeds 0.2 the best known solution is found exactly once. Finally, it is interesting to note that the relative difficulty of the tardiness factor for flow shop instances with a unit penalty is different from the relative difficulty for other instances. That is, the plot in the lower left corner indicates that iterated local search is relatively successful on instances with a tardiness factor bigger than 0.6 whereas in the other plots it is the other way around.

The effect of the tardiness factor on instance difficulty can be explained quite easily. Recall that the tardiness factor indicates how big the distance is between the average due date and the completion time of the last job in the system. A low tardiness factor implies that the average due date is close to the makespan and therefore that many jobs will not be late in most cases. Since these jobs will not be late, their relative order does not matter at all and many solutions will match the best known solution in terms of solution quality. In contrast, if the tardiness factor is large, many jobs will be late and any change in the order of the late jobs may affect the total tardiness of the solution. So, with a large tardiness factor it is much more difficult to find the solution with the lowest total tardiness. Yet, if a unit penalty is applied to each job that is late, then instance with a large tardiness factor are easy again as changes in the order of jobs that are going to be late all the time does not affect the total number of tardy jobs.

6.3 Performance Comparison

Deterministic scheduling problems with tardiness penalties have received a fair share of interest over the years as a wide range of researchers tried to tackle them with dispatching rules, approximate algorithms or exact algorithms. Table 6.2 on the next page lists the main references for applications to the problems that are cov-

Table 6.2: Table of references.

	$1\|\|\sum U_j$	$1\|\|\sum T_j$	$1\|\|\sum w_jU_j$	$1\|\|\sum w_jT_j$	$Pm\|\|\sum U_j$	$Pm\|\|\sum T_j$	$Pm\|\|\sum w_jU_j$	$Pm\|\|\sum w_jT_j$	$Fm\|\text{prmu}\|\sum U_j$	$Fm\|\text{prmu}\|\sum T_j$	$Fm\|\text{prmu}\|\sum w_jU_j$	$Fm\|\text{prmu}\|\sum w_jT_j$
Heuristics	[Jack 55]	[Bake 82, Panw 93]		[Carr 65, Mort 84]		[Bake 73, Dogr 79, Ho 91]		[Koul 93]		[Kim 93a]		
Metaheuristics		[Wilk 71, Fry 89, Pott 91, Hols 92, Baue 00]		[Crau 98, Cong 02, Gros 04, Best 01b, Best 00b, Best 01a, Chan 90, Mats 87]		[Bilg 04, Koul 97, Bean 94, Pavl 03]				[Kim 93a]		[Aden 92]
Exact Algorithms	[Moor 68]	[Elma 68, Shwi 72, Rinn 75, Bake 77, Lawl 77, Pott 82]	[Lawl 69, Croc 98, Szwa 01]	[Pott 85]		[Prit 69, Gupt 73, Aziz 98]				[Kim 93b, Sen 89]		

ered by the benchmark set. Most of these references were adopted from [Koul 94, Table IV]. The list is non-exhaustive in that not all publications of applications are listed. Yet, the lack of references for certain problems does indeed indicate that I am not aware of any application within the row's class of algorithms to the column's type of problem. Also note that there exist some applications to more general problems that are not listed here as they have never been tested on the specific problems covered by the benchmark set. For instance, algorithms have been developed for $Pm|r_j|\sum w_jU_j$ [Seva 01], $1|r_j|\sum w_jU_j$ [Seva 03] and $Pm||\sum E_j + T_j$ [Sivr 99].[3] These problems are outside the scope of this dissertation, but since they can be seen as generalizations of problems within the scope of this dissertation in that $Pm||\sum w_jU_j \propto Pm|r_j|\sum w_jU_j$, $1||\sum w_jU_j \propto 1|r_j|\sum w_jU_j$ and $Pm||\sum T_j \propto Pm||\sum E_j+T_j$, algorithms for these more general problems could handle the more specific problems of the benchmarks set without any need for adjustments.

The empty spaces in Table 6.2 indicate that there are very few applications for problems with a flow shop environment and for problems with a (weighted) unit penalty. For these problems, it is hard to make any comparisons. With respect to the other problems, the multitude of references suggests that there should be ample opportunity for comparisons. Nonetheless, I have been able to collect test results for just a few problem applications. The reason for this is that it would be too time–consuming and cumbersome to reimplement the algorithms based on the description given in the publications alone. Moreover, for most algorithm imple-

[3] Recall that r_j in the problem description indicates that jobs have release dates greater or equal to zero and $E_j + T_j$ indicates that earliness as well as tardiness is penalized.

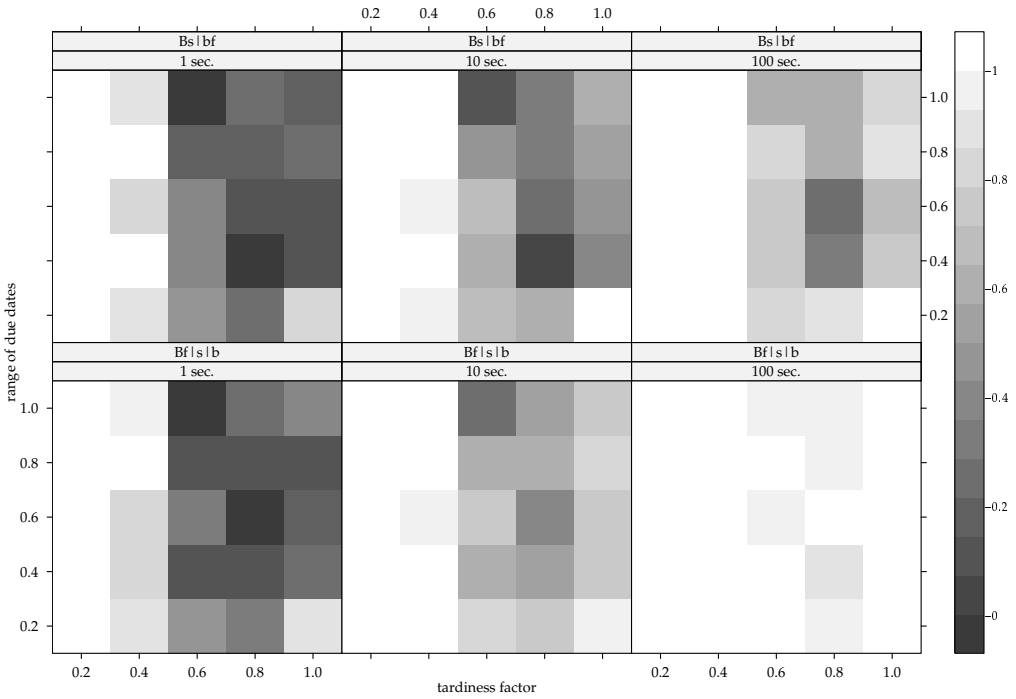


Figure 6.5: Comparison of two iterated local search variants for $1||\sum w_j T_j$.

mentations, it was hard to obtain the source code. Worse still, more often than not, experimental results reported in the publications were based on an unspecified number of runs of the algorithm on an obscure set of irretrievable test instances, while it remained unclear how much tweaking the code had undergone.

The three problems for which I have been able to draw a comparisons between earlier applications and the variant of iterated local search developed within this disseration are $1||\sum U_j$, $1||\sum w_j T_j$ and $Pm||\sum T_j$.

For $1||\sum U_j$ there exists an exact algorithm with complexity $O(n \log n)$ [Moor 68]. This algorithm, which is a *forward* algorithm, is described in detail in [Pine 95]. In the words of [Pine 95, p. 39], the algorithm adds jobs "to a set of on–time jobs in increasing order of due dates. If the inclusion of job j^* results in this job being completed late, the scheduled job with the largest processing time, say job k^*, is marked late and discarded." It has been proven that in this way an optimal schedule for $1||\sum U_j$ is found and so this algorithm was employed to obtain the values of the optimal solutions for the benchmark instances in $1||\sum U_j$. Hence, the plot for $1||\sum U_j$ in Figure 6.2 on page 104 gives an indication of how well iterated local search performs relative to the forward algorithm and, unfortunately, the picture doesn't look bright. That is, for many instances, iterated local search was not able to find a solution of the same quality as the one found by the forward algorithm — even though iterated local search went on for up to 300 seconds whereas the forward algorithm managed to find optimal solutions within a second. Granted,

iterated local search does manage to come quite close to the optimum as a glance at Table B.1 on page 134 will confirm, but apparently it is incredibly difficult for iterated local search to find the last offending job and place in the right position when most jobs have been put in their correct place already.

For $1||\sum w_jT_j$, I had already developed approximate algorithms long before I became aware of the potential of the racing procedure with which iterated local search was developed in Part II [Best 00a]. The approximate algorithm that emerged as the most promising in that early research was a variant of iterated local search where a perturbation of 3 to 12 shift moves was applied to solutions generated by a swap local search followed by a shift local search and where only improving solutions were accepted as new starting solutions [Best 01b]. Figure 6.5 on the preceding page shows how successful that old variant is in finding the optimal solutions for $1||\sum w_jT_j$ and contrasts the performance of the old variant with the performance of the new variant below. The plots below are much paler than the plots above. So, clearly, the new variant outperforms the old variant. Moreover, the plots on the left, which show the variants' performance after one second, are less dissimilar than the plots on the right, which show the variants' performance after 100 seconds. That is, the difference between the two variants becomes more strongly marked with the passing of time. Apparently, all the extra computational effort that went into the racing experiments did pay off. Yet, unfortunately, the performance of the new variant still does not quite match the performance of the state of the art algorithm for $1||\sum w_jT_j$ [Gros 04]. Apparently, experimental effort alone is still no substitute for endless tweaking of code.

For $Pm||\sum T_j$, Michael Pavlin has developed a variant of iterated local search that takes advantage of the implementations of local search that I had developed for $1||\sum w_jT_j$ [Pavl 03]. Michael Pavlin was kind enough to provide the source code for his algorithm. Consequently, it was possible to compare the performance of his algorithm against mine on the same computer. Moreover, the fact that both implementations used the same programming language annulled yet another source of unfair competition [Hook 96].

Figure 6.6 on the next page clearly shows that my algorithm is better. The figure compares the median deviation from the best of both algorithms on $P4||\sum T_j$ and $P2||\sum T_j$ after 1 and 60 seconds. Each point below the dotted lines represents an instance where my algorithm outperforms Pavlin's algorithm and each point above the dotted lines represents an instance where Pavlin's algorithm outperforms mine. Since most points are located below the dotted line, the variant developed for this dissertation "wins". Like in $1||\sum w_jT_j$, the difference between the variants becomes more pronounced with the passing of time.

6.4 Summary

In order to get an impression of its performance, iterated local search has been run ten times for five minutes on a total of 3000 instances. Iterated local search has

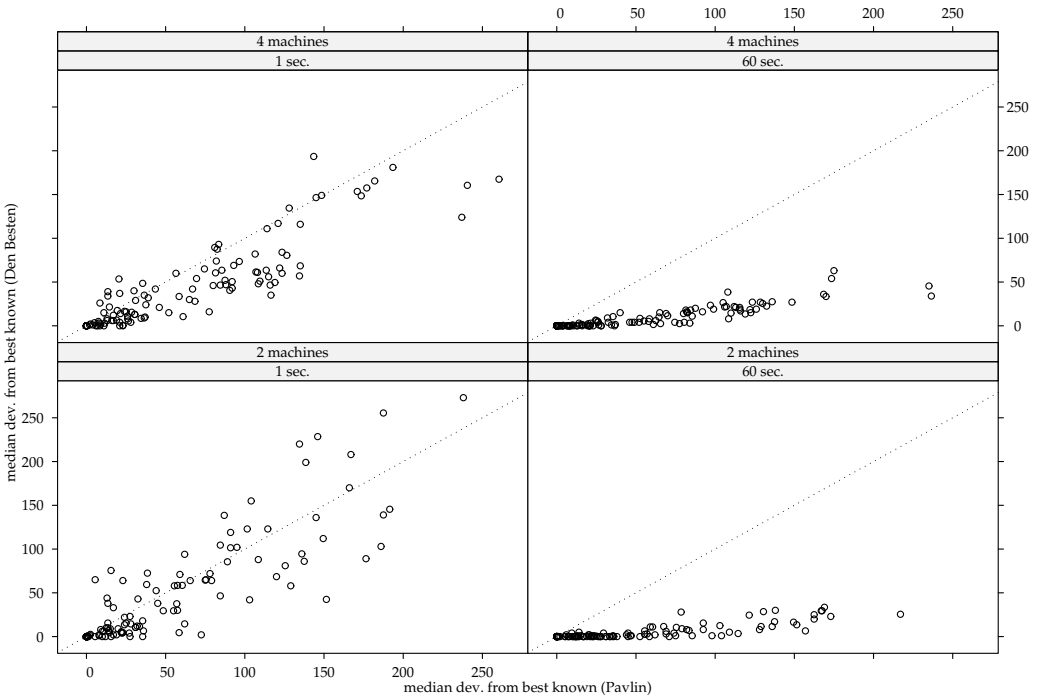


Figure 6.6: Comparison of two iterated local search variants for $Pm||\sum T_j$.

least difficulty with the single machine total weighted tardiness problem and most difficulty with the 100 job 20 machine total weighted tardiness flow shop problem. Also the distribution of the due dates in the instances has a significant effect on the ease with which iterated local search is able to solve them.

It is hard to say how indicate how iterated local search compares to the state of the art due to the limited availability of source codes. In comparison to the two alternative algorithms that were available for testing, the iterated local search developed in this dissertation did seem to do quite well. Yet, iterated local search performs extremely bad on the single machine unit penalty problem, the least complex problem of the pack. For the other problems, the jury is out.

7 Conclusions

The goal of this dissertation was to learn more about the application of approximate algorithms like iterative improvement and iterated local search to scheduling problems with tardiness constraints. Section 7.2 summarizes the lessons that were learnt. Before that, Section 7.1 recapitulates what has been done in the dissertation and afterwards Section 7.3 points out what can be done next.

7.1 Contributions

The contributions of this dissertation are manifold and come in several layers.

Scheduling

At the core is the problem domain from which instances were drawn. The problem domain consists twenty–four distinct instance classes on the basis of twelve distinct problem specifications. Except for the 100 job single machine total weighted tardiness instance class, for each instance class, a new benchmark set of 125 instances was created. Furthermore, solutions were obtained for all 3000 instances and recorded to serve as reference for future research.

In order to find these solutions, new solution representations were developed for the problems with a parallel machine environment. Moreover, well–known speed–up techniques for the single machine total weighted tardiness problem were adapted to the constraints posed by objective functions that ignore weights or impose a unit penalty on each late job. Finally, the speed–up techniques were adapted to work in machine environments with more than one machine in parallel.

The procedures that were employed in the search for solutions were often the first of their kind to be applied to the scheduling problems under investigation. Where a state–of–the–art had been established before, the procedures were able to improve it in many cases and occasionally failed miserable in other cases.

Racing

Acquiring high quality solutions for specific problems was not the only aim of this dissertation. Beyond that, we wanted to investigate trade–offs in the design of search procedures and devise methods for the development of such search procedures.

The main method that was followed for development and design was to carry out racing experiments in which several candidate search procedures are pitted

against each other and the best one is selected on the basis of its performance on a series of test instances. This dissertation contains the results of several hundreds racing experiments representing several weeks of computation time: Six hundred races were carried out to tune the apparent dispatching rule; twenty–four races were carried out to select the best combination of neighborhoods in local search; fourty-eight more were carried out to configure iterated local search. In all these experiments, the test instances were generated on the fly and an effort was made to ensure that they resembled the benchmark instances only at the level of abstract description so that an improper bias towards peculiarities of the benchmark set could be avoided. Furthermore, special care was taken to ensure that the performance assessment in the races reflected the decisions that a sensible developer would make. Whereas the archetypal race only considers the solution quality of the final solutions obtained by the candidates on the test instances, the races for local search also considered the run times required to obtain the solutions and the races for iterated local search considered each pair of solution quality and run time the candidate was capable of producing within the given execution constraints. Many developers are sensible people and so performance assessment on the basis of more than one criterion has been done before. The contribution of this dissertation is to cast the process of development and selection into a mould that can be applied systematically and automatically.

Search and Analysis

This dissertation features the extension and calibration of the apparent urgency dispatching rule to new problems, the first time combination of dispatching rules with insertion heuristics for many problems, the sequencing and selection of neighborhoods for local search out of an unprecedented multitude of candidates, and a systematic consideration of many iterated local search candidates. Without racing, none of it would have been feasible for a single developer. However, not just racing is required for these deeds: In addition, one has to make sense out of the experimental data in one way or another. Survival analysis, hierarchical clustering, and decision tree induction are three such ways. None had been used before for the development of search algorithms and all yielded valuable insights.

7.2 Lessons Learnt

If there is one thing that became clear, then it must be which particular configuration of local search works well on which class of instances. The details of this mapping can be found in the appendix. There are nonetheless also a couple of common patterns that emerge from the experiments.

- Insertion is a powerful heuristic for solution construction whose strength can be enforced greatly by pre–ordering solution components according to their characteristics.

- When faced with a choice between local search neighborhoods, searching several small neighborhoods after one-another is preferable over merging and searching one equivalent monolith.
- Perturbation and acceptance criterion in iterated local search counteract eachother and their specification is relatively independent from the problem domain to which the iterated local search is applied.
- The distribution of values that instantiates a problem if often more important than the high–level problem specification to determine the configuration and performance of a local search algorithm.
- A coarse assessment of performance combining multiple criteria can be a good match to finegrained assessments based on single criteria.

7.3 Future Work

Studies that build upon the research done in this dissertation can go into several directions. First of all, one could extend the problem domain. In addition, one could delve deeper into the study of mechanisms for search. Finally, one could try and improve the procedures for candidate selection and algorithm configuration.

With respect to the problem domain, the first thing that springs to mind is to consider other scheduling problems and other combinatorial optimization problems. What could be even more interesting, however, is to stick with the current problem domain and change the way in which instance data are generated. Rather than from a uniform distribution, the data could be drawn from a normal distribution or a Poisson distribution or attempts could be made to obtain data that reflect real world problems. In [Wats 02, Wats 03] it was found that the structure of problem data has a great impact on algorithm performance. Our experience in this dissertation points in the same direction. It would be interesting to see whether these findings can be substantiated with further evidence.

In relation to search algorithms, it is also tempting to simply repeat the same procedure all over again for different algorithm designs and then to sit back and and watch as racing does its job. Certainly, it is likely that the likes of memetic algorithms, tabu search, and simulated annealing lend themselves just as well to racing as iterated local search although some difficulties might arise due to the fact that not all of them have the level of modularity of iterated local search. But again, a lot of interesting research still remains to be done with regard to iterative improvement, piped local search and iterated local search alone. In relation to iterative improvement, it would be interesting to see whether one could improve performance of first improvement local search by scanning the neighborhood in a random order or by resuming scanning at the point where the latest improvement was found rather than returning to the same starting position after each improvement [Hoos 04]. In relation to piped local search, it would be interesting to see

whether the sequencing of searches that differ in other aspects than just neighborhood definition makes sense [Best 01a]. In addition, it would be interesting to see whether the finding that a sequence of small neighborhoods is better than one big monolith holds for other problem domains. Finally, in relation to iterated local search, it would be interesting to try to pinpoint more precisely how the trade–off between run time and solution quality in local search affects the performance of the iterated local search procedure wrapped around it.

As for racing, there are a number of open issues that need to be resolved which were already discussed in Section 2.1. Clearly, it could be interesting to compare racing with other methods of algorithm configuration. Moreover, we need to improve our understanding of racing as such and explore alternative ways of algorithm performance assessment. Last but not least, we need to find new and better ways to fully exploit the results produced by the racing experiments. Additional methods of analysis, in particular search space analysis [Reev 99], might be helpful in this respect.

So many challenges still lie ahead. Kudos to anyone willing to take them on.

Appendix

A Algorithm Development

A.1 Construction

A.1.1 Apparent Urgency

Table A.1 on the following page tabulates the outcomes of the 600 racing experiments described in Section 3.1. Figure 3.1 on page 51 is based on these results. The goal of the experiments was to determine the best value of look ahead parameter k in apparent urgency. Note however that the value reported in Table A.1 is not k itself but rather $\kappa = \log_2 k$.

Figure A.1 on page 122, Figure A.2 on page 122, and Figure A.3 on page 123 depict the decision tree models for the selection of look ahead parameter k in apparent urgency which was discussed in Section 3.1. The decision trees were generated with the complexity parameter cp set to 6, 8, and 10, respectively. Like in Table A.1, the value on the leaves of the three is κ rather than k. The decision tree of Figure A.3 was actually used to determine value of the look–ahead parameter with which jobs were put in apparent urgency order for the tests reported in Appendix B. In the trees, labels at the branching points indicate what splitting criteria was used. The left branch indicates how to proceed with instance descriptions that fulfill the criterion on the split; the right branch is for those who fail.

A.1.2 Apparent Urgency plus Insertion

Table A.3 on page 124 tabulates for each tested variant of the insertion heuristic how often it has "won" the test. That is, the table indicates how often the variants could come up with the solution with of the highest quality. In case of a draw, if the best solution has been found by several variants, the winner is randomly drawn among them.

Table A.1: Optimal κ values for apparent urgency (I).

TF	RDD	$1\|\|\sum_{j=1}^{100} U_j$	$1\|\|\sum_{j=1}^{100} T_j$	$1\|\|\sum_{j=1}^{100} w_j U_j$	$1\|\|\sum_{j=1}^{100} w_j T_j$	$P2\|\|\sum_{j=1}^{100} U_j$	$P2\|\|\sum_{j=1}^{100} T_j$	$P2\|\|\sum_{j=1}^{100} w_j U_j$	$P2\|\|\sum_{j=1}^{100} w_j T_j$	$P3\|\|\sum_{j=1}^{100} U_j$	$P3\|\|\sum_{j=1}^{100} T_j$	$P3\|\|\sum_{j=1}^{100} w_j U_j$	$P3\|\|\sum_{j=1}^{100} w_j T_j$
0.2	0.2	$2\frac{1}{2}$	$1\frac{1}{2}$	$3\frac{1}{2}$	1	$1\frac{1}{2}$	1	$2\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	2	0
0.4	0.2	$3\frac{1}{2}$	$2\frac{1}{2}$	$3\frac{1}{2}$	2	$2\frac{1}{2}$	$1\frac{1}{2}$	3	1	2	1	$2\frac{1}{2}$	$\frac{1}{2}$
0.6	0.2	4	3	$3\frac{1}{2}$	2	$2\frac{1}{2}$	$1\frac{1}{2}$	3	$1\frac{1}{2}$	$2\frac{1}{2}$	$1\frac{1}{2}$	$2\frac{1}{2}$	1
0.8	0.2	$3\frac{1}{2}$	2	3	$2\frac{1}{2}$	2	1	2	$1\frac{1}{2}$	2	$\frac{1}{2}$	2	1
1	0.2	$1\frac{1}{2}$	-2	2	-2	2	-3	2	-3	2	$-3\frac{1}{2}$	$1\frac{1}{2}$	$-3\frac{1}{2}$
0.2	0.4	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	0	0	-1	0	-1	0	-1	$-\frac{1}{2}$	$-1\frac{1}{2}$
0.4	0.4	$3\frac{1}{2}$	$2\frac{1}{2}$	$3\frac{1}{2}$	$\frac{1}{2}$	$2\frac{1}{2}$	$1\frac{1}{2}$	3	$-\frac{1}{2}$	2	$\frac{1}{2}$	$2\frac{1}{2}$	$-1\frac{1}{2}$
0.6	0.4	$3\frac{1}{2}$	$2\frac{1}{2}$	4	0	3	$1\frac{1}{2}$	3	0	2	1	$2\frac{1}{2}$	$-\frac{1}{2}$
0.8	0.4	$3\frac{1}{2}$	$-\frac{1}{2}$	3	$-\frac{1}{2}$	$2\frac{1}{2}$	$-\frac{1}{2}$	2	$-1\frac{1}{2}$	$2\frac{1}{2}$	$-1\frac{1}{2}$	$1\frac{1}{2}$	-2
1	0.4	$2\frac{1}{2}$	$-1\frac{1}{2}$	$2\frac{1}{2}$	$-1\frac{1}{2}$	1	$-2\frac{1}{2}$	$2\frac{1}{2}$	$-2\frac{1}{2}$	2	$-2\frac{1}{2}$	1	-3
0.2	0.6	$-5\frac{1}{2}$	$-5\frac{1}{2}$	$-5\frac{1}{2}$	$-5\frac{1}{2}$	-6	-6	$-6\frac{1}{2}$	$-6\frac{1}{2}$	$-6\frac{1}{2}$	$-6\frac{1}{2}$	$-7\frac{1}{2}$	$-7\frac{1}{2}$
0.4	0.6	3	$1\frac{1}{2}$	$3\frac{1}{2}$	$-\frac{1}{2}$	2	$\frac{1}{2}$	2	-1	$1\frac{1}{2}$	0	2	$-1\frac{1}{2}$
0.6	0.6	$3\frac{1}{2}$	2	$3\frac{1}{2}$	$-\frac{1}{2}$	$2\frac{1}{2}$	1	3	-1	2	$\frac{1}{2}$	$2\frac{1}{2}$	-1
0.8	0.6	$3\frac{1}{2}$	$-\frac{1}{2}$	$3\frac{1}{2}$	-1	$2\frac{1}{2}$	-1	$2\frac{1}{2}$	$-1\frac{1}{2}$	$2\frac{1}{2}$	$-1\frac{1}{2}$	$2\frac{1}{2}$	-2
1	0.6	4	$-1\frac{1}{2}$	3	$-1\frac{1}{2}$	3	$-1\frac{1}{2}$	$2\frac{1}{2}$	-2	3	$-2\frac{1}{2}$	2	$-2\frac{1}{2}$
0.2	0.8	$-5\frac{1}{2}$	$-5\frac{1}{2}$	-6	-6	$-6\frac{1}{2}$	$-6\frac{1}{2}$	-7	-7	-7	-7	$-7\frac{1}{2}$	$-7\frac{1}{2}$
0.4	0.8	$1\frac{1}{2}$	0	2	$-\frac{1}{2}$	1	-1	$1\frac{1}{2}$	-1	1	-1	1	$-1\frac{1}{2}$
0.6	0.8	$3\frac{1}{2}$	1	$3\frac{1}{2}$	$-\frac{1}{2}$	2	0	3	-1	$1\frac{1}{2}$	$-\frac{1}{2}$	2	$-1\frac{1}{2}$
0.8	0.8	4	0	$3\frac{1}{2}$	-1	$2\frac{1}{2}$	-1	$2\frac{1}{2}$	-2	2	-2	2	-2
1	0.8	4	-1	4	$-1\frac{1}{2}$	$3\frac{1}{2}$	-2	$2\frac{1}{2}$	-2	$2\frac{1}{2}$	-2	$2\frac{1}{2}$	-3
0.2	1	-6	-6	$-6\frac{1}{2}$	$-6\frac{1}{2}$	-7	-7	$-7\frac{1}{2}$	$-7\frac{1}{2}$	$-7\frac{1}{2}$	$-7\frac{1}{2}$	-8	-8
0.4	1	0	0	0	0	0	$-1\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$-1\frac{1}{2}$	$-1\frac{1}{2}$	$\frac{1}{2}$	-2
0.6	1	$3\frac{1}{2}$	0	4	$-\frac{1}{2}$	$2\frac{1}{2}$	$-\frac{1}{2}$	3	$-\frac{1}{2}$	$1\frac{1}{2}$	-1	$2\frac{1}{2}$	$-1\frac{1}{2}$
0.8	1	$3\frac{1}{2}$	$-\frac{1}{2}$	$3\frac{1}{2}$	-1	$3\frac{1}{2}$	-1	3	$-1\frac{1}{2}$	$2\frac{1}{2}$	-2	3	-2
1	1	$4\frac{1}{2}$	$-\frac{1}{2}$	$3\frac{1}{2}$	$-1\frac{1}{2}$	$3\frac{1}{2}$	$-1\frac{1}{2}$	$2\frac{1}{2}$	$-1\frac{1}{2}$	$2\frac{1}{2}$	-2	2	$-2\frac{1}{2}$

Table A.2: Optimal κ values for apparent urgency (II).

TF	RDD	$P4\|\|\sum_{j=1}^{100} U_j$	$P4\|\|\sum_{j=1}^{100} T_j$	$P4\|\|\sum_{j=1}^{100} w_jU_j$	$P4\|\|\sum_{j=1}^{100} w_jT_j$	$F20\|\text{prmu}\|\sum_{j=1}^{50} U_j$	$F20\|\text{prmu}\|\sum_{j=1}^{50} T_j$	$F20\|\text{prmu}\|\sum_{j=1}^{50} w_jU_j$	$F20\|\text{prmu}\|\sum_{j=1}^{50} w_jT_j$	$F20\|\text{prmu}\|\sum_{j=1}^{100} U_j$	$F20\|\text{prmu}\|\sum_{j=1}^{100} T_j$	$F20\|\text{prmu}\|\sum_{j=1}^{100} w_jU_j$	$F20\|\text{prmu}\|\sum_{j=1}^{100} w_jT_j$
0.2	0.2	1	$\frac{1}{2}$	2	0	$-1\frac{1}{2}$	-1	4	-9	-1	-1	5	$-\frac{1}{2}$
0.4	0.2	$1\frac{1}{2}$	1	$2\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	10	$-1\frac{1}{2}$	0	$-\frac{1}{2}$	$5\frac{1}{2}$	$-2\frac{1}{2}$
0.6	0.2	$1\frac{1}{2}$	1	2	1	8	0	3	$\frac{1}{2}$	$\frac{1}{2}$	-1	$4\frac{1}{2}$	$\frac{1}{2}$
0.8	0.2	1	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$3\frac{1}{2}$	4	$1\frac{1}{2}$	$5\frac{1}{2}$	0	3	$-1\frac{1}{2}$
1	0.2	1	-4	$1\frac{1}{2}$	$-3\frac{1}{2}$	-10	-10	-10	-10	-10	-10	-10	-10
0.2	0.4	0	-1	0	-2	-1	$-1\frac{1}{2}$	$4\frac{1}{2}$	-9	-1	-1	$4\frac{1}{2}$	-3
0.4	0.4	$1\frac{1}{2}$	$\frac{1}{2}$	2	-2	0	-2	$5\frac{1}{2}$	$-4\frac{1}{2}$	0	$-\frac{1}{2}$	$4\frac{1}{2}$	$-\frac{1}{2}$
0.6	0.4	2	1	2	$-\frac{1}{2}$	2	-2	$6\frac{1}{2}$	0	1	$-2\frac{1}{2}$	$4\frac{1}{2}$	$-\frac{1}{2}$
0.8	0.4	$1\frac{1}{2}$	$-1\frac{1}{2}$	$1\frac{1}{2}$	$-1\frac{1}{2}$	7	$-\frac{1}{2}$	7	7	$8\frac{1}{2}$	$-\frac{1}{2}$	10	$3\frac{1}{2}$
1	0.4	7	-3	$1\frac{1}{2}$	-4	-10	-10	-10	-10	$4\frac{1}{2}$	$\frac{1}{2}$	$6\frac{1}{2}$	$6\frac{1}{2}$
0.2	0.6	-7	-7	$-7\frac{1}{2}$	$-7\frac{1}{2}$	-1	-2	$4\frac{1}{2}$	1	-1	$-1\frac{1}{2}$	5	-1
0.4	0.6	$1\frac{1}{2}$	0	1	-2	0	$-1\frac{1}{2}$	$5\frac{1}{2}$	$\frac{1}{2}$	0	-2	5	$\frac{1}{2}$
0.6	0.6	$1\frac{1}{2}$	0	2	$-1\frac{1}{2}$	$3\frac{1}{2}$	-3	$9\frac{1}{2}$	2	1	$-1\frac{1}{2}$	9	$\frac{1}{2}$
0.8	0.6	2	$-1\frac{1}{2}$	2	-2	$4\frac{1}{2}$	$\frac{1}{2}$	$8\frac{1}{2}$	7	5	-2	10	3
1	0.6	$1\frac{1}{2}$	$-2\frac{1}{2}$	1	-3	3	3	$3\frac{1}{2}$	$1\frac{1}{2}$	$4\frac{1}{2}$	$3\frac{1}{2}$	$8\frac{1}{2}$	$9\frac{1}{2}$
0.2	0.8	$-7\frac{1}{2}$	$-7\frac{1}{2}$	-8	-8	-1	-3	5	1	$-1\frac{1}{2}$	-2	5	2
0.4	0.8	0	-1	$\frac{1}{2}$	-2	0	$-2\frac{1}{2}$	$6\frac{1}{2}$	-2	0	$-2\frac{1}{2}$	6	$1\frac{1}{2}$
0.6	0.8	$1\frac{1}{2}$	$-\frac{1}{2}$	2	-1	$3\frac{1}{2}$	$-3\frac{1}{2}$	$8\frac{1}{2}$	1	1	$-5\frac{1}{2}$	8	$-\frac{1}{2}$
0.8	0.8	$1\frac{1}{2}$	-2	2	$-2\frac{1}{2}$	5	-2	$8\frac{1}{2}$	$8\frac{1}{2}$	$4\frac{1}{2}$	$-7\frac{1}{2}$	10	3
1	0.8	2	$-2\frac{1}{2}$	3	$-2\frac{1}{2}$	$6\frac{1}{2}$	$\frac{1}{2}$	$6\frac{1}{2}$	$6\frac{1}{2}$	5	-5	8	2
0.2	1	-8	-8	-8	-8	$-1\frac{1}{2}$	$-3\frac{1}{2}$	$4\frac{1}{2}$	$-\frac{1}{2}$	$-5\frac{1}{2}$	-5	$2\frac{1}{2}$	$1\frac{1}{2}$
0.4	1	$-1\frac{1}{2}$	$-2\frac{1}{2}$	$-\frac{1}{2}$	$-1\frac{1}{2}$	0	-3	5	0	0	$-2\frac{1}{2}$	6	2
0.6	1	1	$-1\frac{1}{2}$	2	-2	3	-2	$9\frac{1}{2}$	1	$\frac{1}{2}$	-2	10	3
0.8	1	$1\frac{1}{2}$	-2	$2\frac{1}{2}$	$-2\frac{1}{2}$	3	-10	9	5	3	$-5\frac{1}{2}$	10	4
1	1	$2\frac{1}{2}$	-2	$1\frac{1}{2}$	$-2\frac{1}{2}$	$2\frac{1}{2}$	-10	$6\frac{1}{2}$	8	6	-10	10	-3

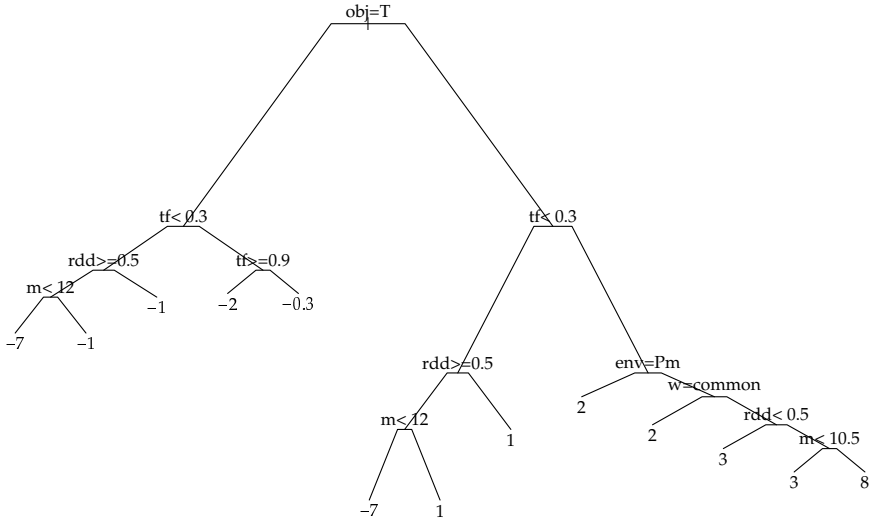


Figure A.1: Decision tree for apparent urgency calibration; $cp = 2^{-6}$

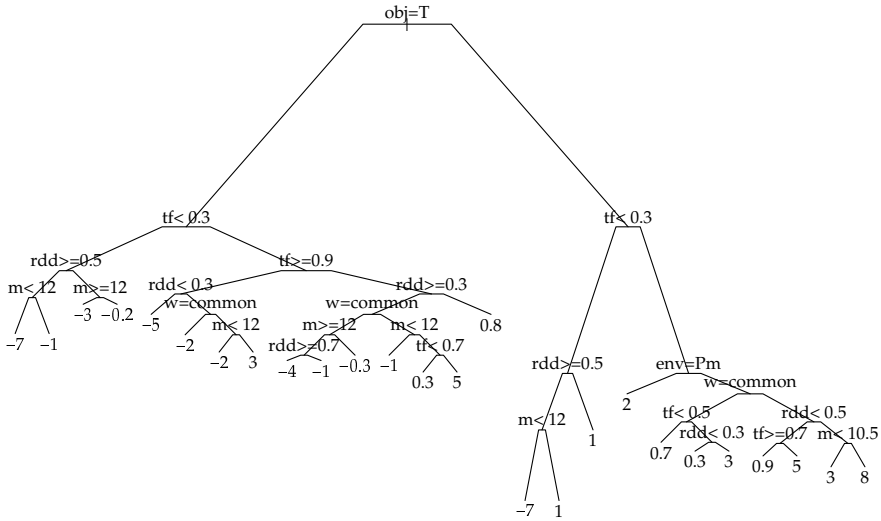


Figure A.2: Decision tree for apparent urgency calibration; $cp = 2^{-8}$

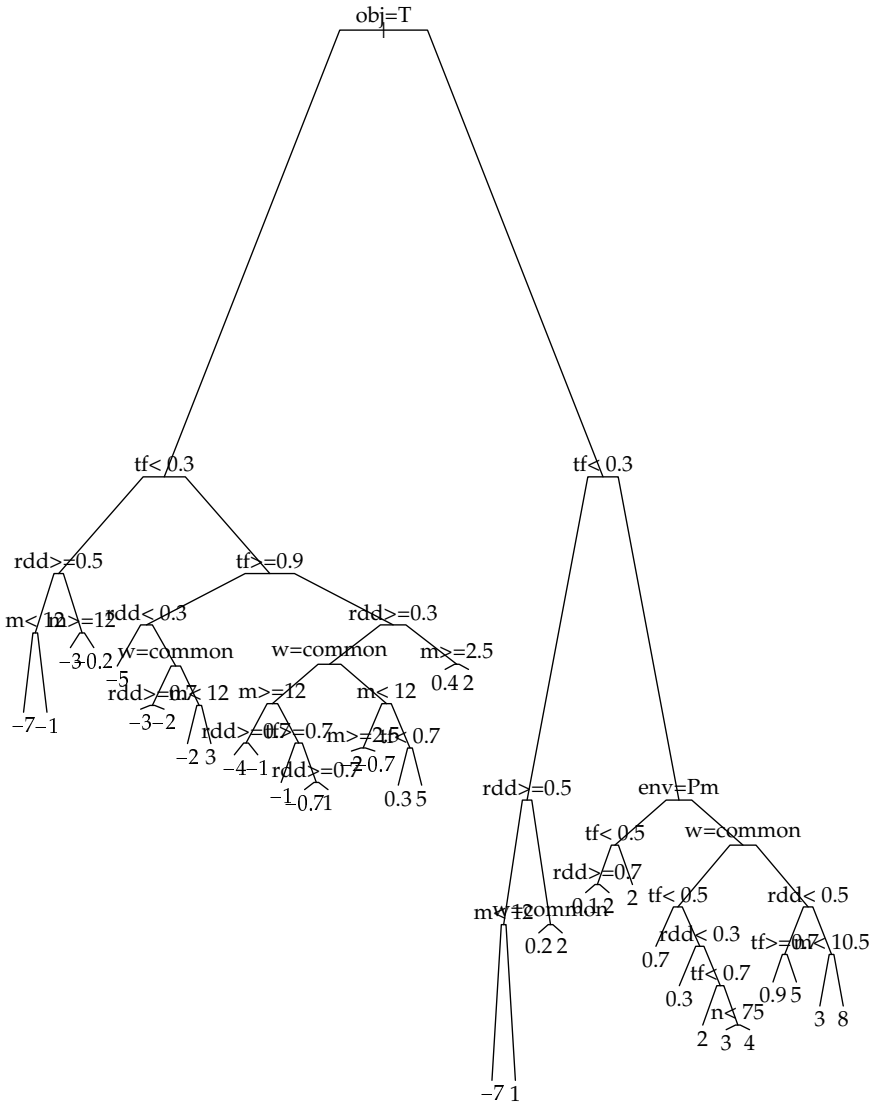


Figure A.3: Decision tree for apparent urgency calibration; $cp = 2^{-10}$

Table A.3: Number of instances "won" by variants of the insertion heurtistic.

	AU	AU|NEH	EDD	EDD|NEH	RND	RND|NEH
$1\|\|\sum_{j=1}^{100} U_j$	18	65	17	103	0	0
$1\|\|\sum_{j=1}^{100} T_j$	19	42	12	119	0	2
$1\|\|\sum_{j=1}^{100} w_j U_j$	14	76	8	62	0	0
$1\|\|\sum_{j=1}^{100} w_j T_j$	10	99	18	58	0	0
$P2\|\|\sum_{j=1}^{100} U_j$	29	119	11	24	0	2
$P2\|\|\sum_{j=1}^{100} T_j$	16	97	15	83	0	7
$P2\|\|\sum_{j=1}^{100} w_j U_j$	25	106	12	28	0	3
$P2\|\|\sum_{j=1}^{100} w_j T_j$	6	110	10	43	0	4
$P3\|\|\sum_{j=1}^{100} U_j$	19	129	12	26	0	7
$P3\|\|\sum_{j=1}^{100} T_j$	16	88	12	67	0	5
$P3\|\|\sum_{j=1}^{100} w_j U_j$	29	115	13	18	0	5
$P3\|\|\sum_{j=1}^{100} w_j T_j$	11	121	9	39	0	2
$P4\|\|\sum_{j=1}^{100} U_j$	20	121	8	25	0	4
$P4\|\|\sum_{j=1}^{100} T_j$	7	85	7	68	0	11
$P4\|\|\sum_{j=1}^{100} w_j U_j$	31	125	8	19	0	3
$P4\|\|\sum_{j=1}^{100} w_j T_j$	9	110	12	45	0	4
$F20\|\text{prmu}\|\sum_{j=1}^{50} U_j$	3	43	2	111	0	28
$F20\|\text{prmu}\|\sum_{j=1}^{50} T_j$	6	32	5	142	2	22
$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j U_j$	0	136	2	57	0	6
$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j T_j$	1	89	3	75	1	8
$F20\|\text{prmu}\|\sum_{j=1}^{100} U_j$	1	68	2	107	0	16
$F20\|\text{prmu}\|\sum_{j=1}^{100} T_j$	2	21	6	162	1	9
$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j U_j$	1	143	2	30	0	2
$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j T_j$	0	83	5	97	0	1

A.2 Local Search

Listed here are the outcomes of the 24 racing experiments of Section 4.2. For every experiment, first a descriptor of corresponding problem class is given and this is followed by a list of variable neighborhood descent descriptions representing those candidates who survived the experiment. The list is ordered according to solution quality, with those configurations yielding the highest quality listed first.

$1||\sum_{j=1}^{100} U_j$ Ffs||; F|f|s; Fs|fs|; F|b|; F||

$1||\sum_{j=1}^{100} T_j$ Ffs|b|; Ff|s|b; Fs|b|; B|s|b; Fs||f; Ff|s|; F|s|; B||s; F||

$1||\sum_{j=1}^{100} w_j U_j$ Bf||fs; Bb|f|s; Bs|b|f; Fbf|fs|; Fb|bf|s; Ff|fs|; Fb|f|s; F|f|s; B|s|; Fb|s|f; F|s|f; F|s|; Fbf||; F|b|bf; F|b|f; F||f; Fb||; F||

$1||\sum_{j=1}^{100} w_j T_j$ F|fs|b; Fbf|s|f; Bf|s|b; Ff|b|s; Ff|s|b; Bf|bf|s; Fs|f|b; Bbf|s|; Bb|f|s; Fs||f; Bs||b; Fbf||; B|f|s; B||s; B||

$P2||\sum_{j=1}^{100} U_j$ F|fs|; Ff|s|f; Fs|b|f; Ff|f|; F||

$P2||\sum_{j=1}^{100} T_j$ Ff|b|s; F|bf|fs; Bf|bf|s; Bf|s|b; Ff|s|b; B|bf|s; Fs||b; Fs|f|; Bf|b|s; Bb|bf|s; Bb|f|s; Bs|f|b; Bb|s|f; Bb|s|; B|s|; Bf||b; Bbf|b|; B|b|f; B|b|; B|f|

$P2||\sum_{j=1}^{100} w_j U_j$ Bb|f|s; Bf|s|; Fb|bf|s; Ff||s; Fs|f|b; Fs||; Ff||b; F|f|; F||

$P2||\sum_{j=1}^{100} w_j T_j$ Ff|bsf|; Ff|bs|; Fb|f|bs; Fbf||s; Ff|b|s; Fb|s|f; Ff|s|b; Ff|bf|s; Ffs||b; Bbf||s; Bf|bf|s; Fs|f|f; Bs|f|b; Bs|b|f; B|f|bs; Bb|bf|s; Bf|b|s; Bf|b|fs; Bb|fs|; Bf|s|b; Bb|s|f; Fs||b; Bs|b|; F|f|s; Bb|f|s; Fs||; B|b|s; B|s|f; Bf||s; B|s|; B|f|b; B|f|bf; Bbf|b|; Bb||bf; B|b|f; Ff||; Bb||; Bf||; F||

$P3||\sum_{j=1}^{100} U_j$ Ff|s|bf; F|s|f; F|f|b; B||

$P3||\sum_{j=1}^{100} T_j$ Ff|b|s; Ff|s|b; Bf|bf|s; B|bf|s; Bf|b|s; Fs||; Bb|bf|s; F|f|s; Bb|f|s; Bb|s|f; Bs|b|; Bb||s; Ff|b|; B|b|f; Bf|b|; B|bf|; F|f|; B||b; B||f; F||

$P3||\sum_{j=1}^{100} w_j U_j$ Fb|f|s; F|bf|s; Bb|f|s; Ff|b|s; F|f|fs; F|f|s; Fs||; Fbf||; Ff||b; F||f; B||

$P3||\sum_{j=1}^{100} w_j T_j$ Fb|f|bs; F|b|fs; Fb|f|fs; Fb|bs|f; Fb|s|f; Ff|s|b; Ff|b|fs; F|fs|b; Ff|b|s; Ff|bf|s; Fs|f|b; Fbf|s|b; Bf|bf|s; Bb|f|s; Fs|f|; Bb|bf|s; Ffs||; B|bf|s; F|f|s; Bf|b|s; Bb|s|f; B|b|fs; F|s|; Bb|s|; B|s|; Ff|b|; B|bf|; Bb||bf; Bb||f; Bf||b; F|f|; B||b; B||f; F||

$P4||\sum_{j=1}^{100} U_j$ Fs|bf|; Fs||; F||f; B||

$P4||\sum_{j=1}^{100} T_j$ Ff|bf|s; Fb|f|bs; Fb|f|s; Ff|b|s; Ff|s|b; Bf|bf|s; F|s|b; Fs||; Bb|s|f; Bb|bf|s; Bb|f|s; Bf|b|s; F|f|s; Ff|b|; Bf|b|; Bb||f; Bb||; Bf||; F||

$P4||\sum_{j=1}^{100} w_j U_j$ Bb|f|s; Ff|bf|fs; Fb|f|bs; Ff||bs; Ff||s; Fs|f|b; Fbf||; F|f|b; F|f|f

$P4||\sum_{j=1}^{100} w_j T_j$ Ff|bs|; Fbsf||; Ff|b|fs; Fs|b|f; Fbf|s|; Fb|f|s; Ff|s|b; Ff|b|s; F|s|; Bf|bf|s; Bb|bf|s; Bb|f|bs; Bb||fs; Bb|f|s; Bb|s|f; Bf|b|s; Bb||s; Ff||b; F|f|bf; Bf|bf|; B|f|b; B||bf; Bb||f; F||f; F||

$F20|\text{prmu}|\sum_{j=1}^{50} U_j$ F|s|f; F||s; F||f; F||b; F||

$F20|\text{prmu}|\sum_{j=1}^{50} T_j$ Fbsf||; Fbf||; Fbs|f|b; Fbs|f|; F|bs|f; F|s|bf; Fb|bsf|; Fs|fs|b; Fb|fs|bf; Fs|f|b; Fb|s|bf; F|fs|b; Fs|b|f; Fb|f|bs; Ff||bsf; Fb|f|s; Fb|f|bf; Ff|b|s; F|s|f; Ff|s|bf; F|b|f; Fb||bf; Ff|bf|s; Ff|s|b; Ff||b; Fs|b|; Bs||bf; Bs|fs|b; Bs|f|b; Bfs||b; Bb|s|f; Bb|s|fs; Bfs|b|f; Bf|s|bf; Bs|b|bf; Bs|b|f; F|s|; Bf|s|b; Bbf|s|; B|bf|s; Bf|b|bsf; Bf|bs|; B|f|bsf; Bf|fs|b; Bf|b|fs; Bf|b|s; Bs||f; Bf|bf|s; Bs|fs|; Bb|f|bs; Ff|s|; Bb|f|s; Bf|b|; B|bf|; Bb||f; Ff||; B|f|s; Bs||b; B||s; Bf||; B|b|; B||

$F20|\text{prmu}|\sum_{j=1}^{50} w_j U_j$ Fbf||s; Ff|bsf|; Bs|f|; Fb|f|s; Bf||s; Fs||f; B|s|; F|s|; F|bf|; F||f; Fb||; B||

$F20|\text{prmu}|\sum_{j=1}^{50} w_jT_j$ Fbsf||; Fbf||s; F|bf|s; F|bf|; Fb|bsf|; F|b|bsf; Fs|bf|; Fs|f|b; Fs|b|f; Fb|f|bsf; Fb||fs; F|s|f; Ffs||bsf; F|fs|bsf; Fb|f|bf; Ffs|b|; F|fs|b; Ff|s|bf; Ff|s|bsf; Ff|b|bsf; Ff|s|bs; F|f|bsf; Ff|bf|; F|s|b; Ff|b|fs; Ff|b|s; Ff|s|b; Bs|f|bsf; Bs|f|bs; F||s; Bb|bs|bf; Bs|b|bf; Bs|fs|b; Ff||b; F|f|b; Bs|b|bsf; B|bsf|; Bs|b|f; Bs|b|fs; B|b|fs; Bs|f|b; Bb|bs|f; B|s|bf; Bfs|b|; Bfs|b|f; Bf|b|bs; Bf|b|bsf; B|f|bs; Bf|s|b; Bf|b|fs; Bf|fs|bs; Bf|fs|bf; Bf|s|bs; Bf|b|s; Bf|bf|s; Bf|fs|b; Bb|f|bs; Bf|s|bf; Bs|fs|; Bb|f|s; Bs||f; B|s|f; B|f|bf; Bf||b; Bb|f|; Bb||f; Bs||b; Bb||s; Bf|s|; F||f; B||s; B||f; B||

$F20|\text{prmu}|\sum_{j=1}^{100} U_j$ Ff|s|; Fs||f; F||s; F||f; F||b; F||

$F20|\text{prmu}|\sum_{j=1}^{100} T_j$ Fbsf||; Fs|b|bsf; Fs|bs|fs; Fs|f|bsf; Fs|bsf|; Fs|bf|; F|s|bf; Fs|f|bs; Fs|bs|f; Fs|b|f; Fs|b|bf; Fs|f|b; Fs|fs|bf; Fs|f|bf; Fs||f; F|s|f; Fs|b|; Fs||; Ff|bsf|; Ff|s|bsf; Ff|s|bs; Ff|fs|bsf; Ff|bs|fs; Ff||bs; Ff|bf|s; Ff|s|bf; Ff|b|fs; Ff|b|bs; Ff|b|s; Ff|bf|; F|f|bf; Ff|fs|b; Ff|s|b; Ff||b; Bs|b|fs; Bs|b|bsf; Bbs|f|bsf; Bs|bs|fs; Bs||bsf; Bbs|f|bf; Bbs||bf; B|bs|bf; Bbs|bsf|; Bbs|fs|b; Bbs||bsf; B|bs|bsf; Bbs||fs; B|bs|fs; Bbs||f; Bs|bs|bf; Bbs|f|b; Bs|b|f; Bs|bf|; Bs|bs|f; Bs|f|bsf; Ff|s|; Ff||s; Bs|b|bf; Bb|s|fs; Bb|bs|bf; Bf|s|b; Bs|f|bs; Bs|fs|b; Bf|bs|; Bf|s|bs; Bf|fs|bs; Bs|f|bf; Bb|bs|f; Bf|bs|bf; Bf|fs|b; Bf|b|fs; Bf|fs|bsf; Bf|s|bf; Bfs|b|; Bf|bsf|; Bb|s|f; Bs|f|b; Bf|b|bs; Ff||; Bf|b|s; Bf|bf|s; Bb|f|bs; Bb|s|bsf; Bf|fs|bf; Bf|b|bsf; Bb|f|s; Bs|fs|; Bs||f; Bf|bf|; Bb|f|; Bf|b|bf; Bf|b|; Bs||b; Bf|s|; Bs|bs|; Bf|fs|; Bs||; Bf||; Bb||; B||

$F20|\text{prmu}|\sum_{j=1}^{100} w_jU_j$ Bb|f|fs; Fb|bf|s; F|bf|s; Bb|f|s; B|f|s; F||fs; F|f|s; F|b|fs; Bs||f; Bb|s|f; F|s|f; B|s|; F|s|; Fbf||; F|b|bf; Fb|f|; F|f|; F|b|

$F20|\text{prmu}|\sum_{j=1}^{100} w_jT_j$ Fs|fs|b; Fs|bsf|; Fs|f|bsf; Fs||bf; Fs|b|fs; Fbsf||; Fs|f|b; Fs|b|bf; Fs|b|f; Fbs||bf; Fbs|fs|; F|bs|fs; Fbs|fs|b; Fs|fs|; Fbf|s|; F||bf; Fs|f|; Fs|b|; Ffs|bsf|; Fs||; Ff|bsf|; Ff|s|bsf; Ff|b|bsf; Ff|fs|bsf; Ff|bs|bsf; Ff|b|fs; Ff||bs; Ff|b|bs; Ff|s|bf; Ff|b|s; Ff|s|bs; Ff|fs|bs; Ff|fs|bf; Ff||bf; Ff|s|b; Ff|b|bf; Ff|fs|b; Ff|b|; Bs|bs|bsf; Bs|bs|fs; Bs|b|bf; Bs|b|fs; Bs|bf|; Bf|bs|; Bs|f|bf; Bs|bs|bf; Bs|f|b; Bs|bs|f; Bs|f|bs; Bbs|fs|bf; Bf|bs|bf; Bb|s|fs; Bb|s|bsf; Bf|fs|b; Bf|b|fs; Bf|fs|bsf; Bf|s|bf; Bfs|b|; Bf|bsf|; Bb|s|f; Bs|f|b; Bf|b|bs; Bb|s|bf; Bbs|bsf|; B|bs|bsf; Bf|bf|bs; Bb|s|fs; Bb|s|bsf; Bfs|b|s; Bf|s|bf; Bbs|bf|; Bf|b|bs; Bb|s|f; Bf|b|fs; Bf|b|s; Bs||fs; B|f|b; Bf|b|bf; Bs||f; Bs||bs; Bs|b|; B||s; B|f|; B||

A.3 Iterated Local Search

A.3.1 Preliminary Race

Below is a list of iterated local search candidates that were selected in the races described in Section 5.2. In addition, tabulation of the features of these candidates is given in Table A.4 on the facing page. In the list below, the candidates are named according to the following scheme:$L\lambda P\tau\sigma A\alpha$, where λ is the local search employed ($\lambda \equiv \emptyset$ indicates the absense of local search), τ and σ are the type and number of kicks in the pertubation phase, and α is the number of non–improving solutions that are accepted before backtracking to the most currently best solution. The candidates are ordered according to their ϵ–indicator score.

$1||\sum_{j=1}^{100} U_j$ LFf|s|bPb5A10; LFf|s|bPf1A10; LFf|s|bPf5A10; LFf|s|bPf10A0; LFf|s|bPf1A5; LFf|s|bPf5A5; LFf|s|bPs1A5; LFf|s|bPs5A10; LFf|s|bPs1A10; LFf|s|bPf10A5; LFf|s|bPs5A5; LFf|s|bPb10A5

$1||\sum_{j=1}^{100} T_j$ LBb|f|sPf5A0; LFf|s|bPb1A5; LBb|f|sPf5A5; LBb|f|sPs5A10; LBb|f|sPs1A5; LFf|s|bPb1A10; LFf|s|bPb5A5; LFf|s|bPs1A10; LFf|s|bPb10A0; LFf|s|bPs1A0; LBb|f|sPb1A10; LBb|f|sPb1A5; LBb|f|sPs1A10; LBb|f|sPf10A0; LBb|f|sPf10A5; LBb|f|sPf10A10; LFf|s|bPf1A0; LFf|s|bPb1A0; LBb|f|sPs1A0; LBb|f|sPf5A10; LBb|f|sPs10A5; LBb|f|sPs10A10; LFf|s|bPs1A5; LBb|f|sPs10A0; LBb|f|sPs5A0; LBb|f|sPb10A10; LFf|s|bPb10A5; LBb|f|sPs5A5; LBb|f|sPb10A0; LFf|s|bPf1A5; LFf|s|bPb10A10

$1||\sum_{j=1}^{100} w_jU_j$ LFf|s|bPs1A5; LFf|s|bPs1A10; LFf|s|bPb5A5; LFf|s|bPb10A0; LBb|f|sPs1A5; LFf|s|bPb5A0; LBb|f|sPs1A10; LBb|f|sPs1A0; LBb|f|sPs5A5; LFf|s|bPb5A10

Table A.4: Component count of survivors per race.

	local search			permutation						acceptance		
	Bb|f|s	Ff|s|b	$\emptyset$	s	b	f	1	5	10	0	5	10
$1\|\|\sum_{j=1}^{100} U_j$	0	**12**	0	4	2	**6**	4	**5**	3	1	**6**	5
$1\|\|\sum_{j=1}^{100} T_j$	**19**	12	0	**12**	11	8	**13**	7	11	10	**11**	10
$1\|\|\sum_{j=1}^{100} w_j U_j$	4	**6**	0	**6**	4	0	**5**	4	1	3	**4**	3
$1\|\|\sum_{j=1}^{100} w_j T_j$	**17**	14	0	9	**12**	10	**14**	11	6	**12**	9	10
$P2\|\|\sum_{j=1}^{100} U_j$	27	27	27	27	27	27	27	27	27	27	27	27
$P2\|\|\sum_{j=1}^{100} T_j$	**23**	17	0	**14**	**14**	12	15	**16**	9	12	**14**	**14**
$P2\|\|\sum_{j=1}^{100} w_j U_j$	**15**	11	0	**11**	9	6	**13**	11	2	**12**	6	8
$P2\|\|\sum_{j=1}^{100} w_j T_j$	9	**11**	0	**9**	8	3	**12**	5	3	7	**9**	4
$P3\|\|\sum_{j=1}^{100} U_j$	15	**26**	0	13	**18**	10	14	**15**	12	**14**	13	**14**
$P3\|\|\sum_{j=1}^{100} T_j$	7	**8**	0	**9**	5	1	**7**	4	4	**10**	3	2
$P3\|\|\sum_{j=1}^{100} w_j U_j$	**23**	20	0	14	**18**	11	**15**	14	14	**15**	**15**	13
$P3\|\|\sum_{j=1}^{100} w_j T_j$	**12**	7	0	**8**	**8**	3	**13**	5	1	**9**	5	5
$P4\|\|\sum_{j=1}^{100} U_j$	6	**10**	0	3	**7**	6	**8**	5	3	**7**	5	4
$P4\|\|\sum_{j=1}^{100} T_j$	**16**	14	0	**11**	9	10	**15**	10	5	**14**	8	8
$P4\|\|\sum_{j=1}^{100} w_j U_j$	**5**	3	0	3	**4**	1	**4**	3	1	**4**	2	2
$P4\|\|\sum_{j=1}^{100} w_j T_j$	**6**	**6**	0	4	3	**5**	**8**	3	1	**6**	3	3
$F20\|\text{prmu}\|\sum_{j=1}^{50} U_j$	6	**18**	0	**9**	8	7	**12**	10	2	8	**9**	7
$F20\|\text{prmu}\|\sum_{j=1}^{50} T_j$	0	**7**	0	1	**3**	**3**	**7**	0	0	**3**	2	2
$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j U_j$	6	**13**	0	**9**	4	6	2	**12**	5	**11**	4	4
$F20\|\text{prmu}\|\sum_{j=1}^{50} w_j T_j$	7	**9**	0	6	**7**	3	**13**	3	0	**6**	5	5
$F20\|\text{prmu}\|\sum_{j=1}^{100} U_j$	0	**4**	0	0	0	**4**	**2**	**2**	0	1	1	**2**
$F20\|\text{prmu}\|\sum_{j=1}^{100} T_j$	9	**11**	0	6	**9**	5	**16**	2	2	6	**8**	6
$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j U_j$	5	**20**	0	**11**	3	**11**	5	**12**	8	8	8	**9**
$F20\|\text{prmu}\|\sum_{j=1}^{100} w_j T_j$	14	**15**	0	8	10	**11**	**15**	10	4	**13**	8	8

$1||\sum_{j=1}^{100} w_j T_j$ LBb|f|sPf10A5; LBb|f|sPs1A10; LBb|f|sPf5A10; LBb|f|sPf10A10; LBb|f|sPf10A0; LFf|s|bPb1A10; LBb|f|sPb1A10; LFf|s|bPb5A10; LBb|f|sPs5A0; LFf|s|bPb5A0; LFf|s|bPb1A5; LFf|s|bPf1A10; LFf|s|bPf1A5; LFf|s|bPf1A0; LBb|f|sPf5A5; LBb|f|sPf5A0; LFf|s|bPb5A5; LBb|f|sPb5A10; LFf|s|bPs1A5; LBb|f|sPb1A0; LBb|f|sPb1A5; LBb|f|sPs10A5; LBb|f|sPs10A0; LBb|f|sPs1A5; LFf|s|bPb10A0; LFf|s|bPb1A0; LBb|f|sPb5A0; LBb|f|sPs1A0; LFf|s|bPs1A10; LFf|s|bPf5A0; LFf|s|bPs5A10

$P2||\sum_{j=1}^{100} U_j$ LFf|s|bPs1A5; LFf|s|bPb10A10; LFf|s|bPb1A10; LFf|s|bPb5A0; LFf|s|bPb10A0; LFf|s|bPf1A10; LFf|s|bPs1A10; LFf|s|bPf10A5; LBb|f|sPb1A10; LFf|s|bPb10A5; LBb|f|sPs5A0; LBb|f|sPs10A0; LFf|s|bPf1A5; LFf|s|bPf5A5; LFf|s|bPb5A5; LBb|f|sPb5A5; LFf|s|bPb5A10; LFf|s|bPs1A0; LFf|s|bPf5A0; LFf|s|bPs10A10; LFf|s|bPs10A0; LBb|f|sPb10A0; LFf|s|bPf5A10; LFf|s|bPs10A5; LBb|f|sPb5A0; LFf|s|bPs5A10; LBb|f|sPf5A10; LBb|f|sPb1A5; LFf|s|bPs5A0; LFf|s|bPf10A0; LFf|s|bPb1A5; LFf|s|bPb1A0; LFf|s|bPf10A10; LBb|f|sPf5A5; LFf|s|bPs5A5; LBb|f|sPs1A5; LBb|f|sPf10A0; LBb|f|sPs5A5; LBb|f|sPb1A0; LBb|f|sPs1A10; LPf1A0; LPs5A0; LPf5A0; LPs10A0; LPb10A0; LPf10A0; LPf1A5; LPs5A5; LPf5A5; LPs10A5; LPf10A5; LPf1A10; LPs5A10; LPf5A10; LPs10A10; LPb10A10; LPf10A10; LBb|f|sPf5A0; LBb|f|sPb10A10; LPb1A0; LBb|f|sPf10A10; LPb5A0; LBb|f|sPb5A10; LBb|f|sPb10A5; LFf|s|bPf1A0; LBb|f|sPs5A10; LPb1A10; LBb|f|sPf10A5; LBb|f|sPf1A5; LBb|f|sPs1A0; LBb|f|sPf1A10; LBb|f|sPf1A0; LBb|f|sPs10A5; LPs1A0; LBb|f|sPs10A10; LPb5A5; LPb10A5; LPb5A10; LPs1A10; LPs1A5; LPb1A5

$P2||\sum_{j=1}^{100} T_j$ LBb|f|sPs5A0; LFf|s|bPs5A5; LFf|s|bPb1A10; LBb|f|sPs10A0; LFf|s|bPs1A5; LFf|s|bPf1A5; LFf|s|bPb1A5; LBb|f|sPf5A5; LBb|f|sPs1A5; LBb|f|sPf5A10; LFf|s|bPb1A0; LFf|s|bPb5A5; LBb|f|sPs1A10; LBb|f|sPf10A10; LBb|f|sPf1A0; LBb|f|sPb1A0; LBb|f|sPs1A0; LBb|f|sPs10A10; LBb|f|sPf10A0; LBb|f|sPs5A5; LFf|s|bPs5A10; LFf|s|bPs1A10; LFf|s|bPb5A0; LFf|s|bPf1A10; LFf|s|bPs5A0; LBb|f|sPf1A10; LBb|f|sPs5A10; LFf|s|bPb10A5; LBb|f|sPf10A5; LBb|f|sPb5A10; LFf|s|bPb5A10; LBb|f|sPb1A5; LFf|s|bPb10A0; LBb|f|sPb5A0; LBb|f|sPf5A0; LBb|f|sPb5A5; LBb|f|sPs10A5; LBb|f|sPf1A5; LFf|s|bPb10A10; LFf|s|bPf5A10

$P2||\sum_{j=1}^{100} w_j U_j$ LBb|f|sPs1A5; LFf|s|bPb10A0; LBb|f|sPb5A5; LBb|f|sPb5A0; LBb|f|sPs5A0; LBb|f|sPs1A0; LFf|s|bPb5A10; LBb|f|sPs1A10; LBb|f|sPb1A0; LFf|s|bPs1A0; LFf|s|bPb5A0; LBb|f|sPb1A10; LFf|s|bPf1A5; LFf|s|bPb1A10; LFf|s|bPf1A0; LBb|f|sPf5A0; LFf|s|bPs1A5; LBb|f|sPs5A5; LBb|f|sPs10A0; LBb|f|sPf1A0; LBb|f|sPs5A10; LBb|f|sPb5A10; LBb|f|sPf5A5; LFf|s|bPs1A10; LFf|s|bPf1A10; LFf|s|bPs5A0

$P2||\sum_{j=1}^{100} w_j T_j$ LBb|f|sPb1A0; LBb|f|sPs1A10; LFf|s|bPb1A0; LBb|f|sPs5A0; LFf|s|bPs5A0; LFf|s|bPb1A5; LFf|s|bPb5A5; LBb|f|sPf5A5; LFf|s|bPs1A5; LBb|f|sPs1A5; LFf|s|bPs1A0; LBb|f|sPf10A5; LBb|f|sPs10A0; LBb|f|sPs1A0; LFf|s|bPf1A5; LFf|s|bPb5A10; LFf|s|bPs1A10; LFf|s|bPb1A10; LFf|s|bPb10A5; LBb|f|sPb1A5

$P3||\sum_{j=1}^{100} U_j$ LFf|s|bPb5A0; LFf|s|bPf5A0; LFf|s|bPs5A0; LFf|s|bPf10A10; LFf|s|bPb5A5; LFf|s|bPf5A5; LFf|s|bPf1A5; LFf|s|bPs1A10; LBb|f|sPb5A0; LFf|s|bPb10A0; LFf|s|bPf5A10; LFf|s|bPb10A5; LFf|s|bPb10A10; LFf|s|bPb1A10; LFf|s|bPf1A0; LFf|s|bPs5A5; LBb|f|sPb10A0; LFf|s|bPf1A10; LFf|s|bPb5A10; LFf|s|bPf10A5; LBb|f|sPs1A10; LFf|s|bPb1A0; LBb|f|sPs5A0; LFf|s|bPb1A5; LFf|s|bPf10A0; LFf|s|bPs5A10; LFf|s|bPs1A5; LFf|s|bPb1A5; LBb|f|sPb10A10; LFf|s|bPs10A10; LFf|s|bPs10A0; LBb|f|sPs5A0; LFf|s|bPs1A0; LBb|f|sPb5A10; LBb|f|sPb10A5; LBb|f|sPs1A5; LBb|f|sPb5A5; LBb|f|sPb1A5; LBb|f|sPf5A0; LBb|f|sPs10A5; LBb|f|sPb1A10

$P3||\sum_{j=1}^{100} T_j$ LBb|f|sPb5A0; LFf|s|bPs1A0; LFf|s|bPs1A5; LBb|f|sPf10A0; LBb|f|sPs1A0; LBb|f|sPs1A5; LBb|f|sPb10A0; LFf|s|bPb1A0; LBb|f|sPs5A0; LFf|s|bPb5A0; LBb|f|sPs1A10; LFf|s|bPs10A0; LFf|s|bPb1A5; LFf|s|bPs10A10; LFf|s|bPs5A0

$P3||\sum_{j=1}^{100} w_j U_j$ LBb|f|sPs5A0; LBb|f|sPs1A5; LFf|s|bPb1A5; LFf|s|bPs1A5; LBb|f|sPb1A10; LFf|s|bPf1A0; LBb|f|sPb5A0; LFf|s|bPs1A0; LBb|f|sPb5A5; LBb|f|sPs1A10; LBb|f|sPb10A5; LBb|f|sPs1A0; LBb|f|sPs5A10; LBb|f|sPb10A0; LBb|f|sPs10A0; LBb|f|sPf5A5; LFf|s|bPb10A10; LFf|s|bPb10A5; LBb|f|sPs5A5; LBb|f|sPb5A10; LFf|s|bPb1A10; LFf|s|bPb1A0; LBb|f|sPb5A10; LFf|s|bPb5A10; LFf|s|bPs10A0; LFf|s|bPb1A0; LFf|s|bPb5A5; LBb|f|sPb10A10; LBb|f|sPf5A10; LBb|f|sPb1A0; LFf|s|bPb5A0; LBb|f|sPb1A5; LFf|s|bPs1A10; LBb|f|sPs10A5; LFf|s|bPf10A0; LFf|s|bPs5A0; LFf|s|bPf1A10; LBb|f|sPf10A5; LBb|f|sPf10A10; LFf|s|bPf5A0; LFf|s|bPf5A10; LBb|f|sPf1A5; LFf|s|bPs10A5; LFf|s|bPf10A5

$P3||\sum_{j=1}^{100} w_j T_j$ LBb|f|sPs1A0; LBb|f|sPf5A0; LBb|f|sPb5A0; LFf|s|bPf1A0; LBb|f|sPs1A10; LBb|f|sPb1A0; LBb|f|sPb5A5; LBb|f|sPs1A5; LBb|f|sPb1A10; LBb|f|sPb1A5; LFf|s|bPs1A0; LFf|s|bPs1A5; LBb|f|sPs5A10; LFf|s|bPb1A0; LBb|f|sPs5A0; LFf|s|bPb1A10; LBb|f|sPb10A0; LFf|s|bPs1A10; LFf|s|bPf1A5

$P4||\sum_{j=1}^{100} U_j$ LBb|f|sPb5A0; LFf|s|bPf5A0; LBb|f|sPb1A10; LFf|s|bPb10A0; LFf|s|bPf1A0; LBb|f|sPs5A0; LBb|f|sPs1A5; LBb|f|sPb10A0; LFf|s|bPb1A5; LBb|f|sPb1A5; LFf|s|bPf5A5; LFf|s|bPb5A0; LFf|s|bPs1A10; LFf|s|bPf1A10; LFf|s|bPf1A5; LFf|s|bPf10A10

$P4||\sum_{j=1}^{100} T_j$ LFf|s|bPs1A0; LFf|s|bPb5A0; LFf|s|bPb1A0; LFf|s|bPf1A0; LBb|f|sPb1A0; LFf|s|bPf1A10; LBb|f|sPs5A0; LFf|s|bPb1A10; LFf|s|bPs5A0; LBb|f|sPf5A5; LBb|f|sPs1A10; LBb|f|sPs1A5; LBb|f|sPb1A5; LBb|f|sPf10A0; LFf|s|bPs1A10; LBb|f|sPb5A0; LBb|f|sPf5A0; LBb|f|sPb10A0; LBb|f|sPs1A0; LBb|f|sPs5A5; LBb|f|sPf10A10; LFf|s|bPb5A5; LFf|s|bPs1A5; LBb|f|sPs10A10; LFf|s|bPf1A5; LBb|f|sPf1A0; LFf|s|bPs10A10; LFf|s|bPb1A5; LBb|f|sPf5A10; LFf|s|bPf5A0

$P4||\sum_{j=1}^{100} w_jU_j$ LBb|f|sPb5A0; LBb|f|sPs5A0; LBb|f|sPb5A5; LBb|f|sPb1A10; LBb|f|sPs1A10; LFf|s|bPb10A0; LFf|s|bPf1A0; LFf|s|bPs1A5

$P4||\sum_{j=1}^{100} w_jT_j$ LBb|f|sPs1A0; LBb|f|sPf5A0; LBb|f|sPs1A5; LBb|f|sPb5A0; LFf|s|bPs1A5; LBb|f|sPf10A0; LFf|s|bPs1A10; LFf|s|bPf1A10; LFf|s|bPb1A0; LFf|s|bPf1A0; LFf|s|bPb1A5; LBb|f|sPf5A10

$F20|\text{prmu}|\sum_{j=1}^{50} U_j$ LFf|s|bPf1A0; LFf|s|bPf5A10; LFf|s|bPs1A5; LFf|s|bPf1A10; LFf|s|bPf5A0; LFf|s|bPf1A5; LFf|s|bPf5A5; LFf|s|bPs10A0; LFf|s|bPs5A0; LFf|s|bPs5A5; LBb|f|sPb1A10; LFf|s|bPs1A10; LFf|s|bPb1A0; LFf|s|bPb1A5; LFf|s|bPb5A0; LFf|s|bPb1A10; LFf|s|bPs1A0; LFf|s|bPb5A5; LFf|s|bPs10A10; LBb|f|sPb1A5; LBb|f|sPs1A5; LBb|f|sPs5A10; LBb|f|sPf5A5; LBb|f|sPb5A0

$F20|\text{prmu}|\sum_{j=1}^{50} T_j$ LFf|s|bPf1A0; LFf|s|bPb1A5; LFf|s|bPf1A5; LFf|s|bPb1A10; LFf|s|bPs1A0; LFf|s|bPb1A0; LFf|s|bPf1A10

$F20|\text{prmu}|\sum_{j=1}^{50} w_jU_j$ LFf|s|bPf5A5; LBb|f|sPs10A0; LFf|s|bPb5A10; LFf|s|bPf1A0; LBb|f|sPs5A10; LFf|s|bPs5A5; LFf|s|bPf10A0; LBb|f|sPf5A0; LFf|s|bPf5A0; LFf|s|bPb5A0; LBb|f|sPs5A0; LFf|s|bPs10A0; LFf|s|bPs5A10; LBb|f|sPs10A10; LFf|s|bPb10A0; LFf|s|bPs1A0; LFf|s|bPs5A0; LFf|s|bPb5A5; LBb|f|sPf5A5

$F20|\text{prmu}|\sum_{j=1}^{50} w_jT_j$ LFf|s|bPb1A5; LBb|f|sPb1A5; LFf|s|bPs1A5; LFf|s|bPb1A10; LFf|s|bPs1A10; LFf|s|bPf1A5; LBb|f|sPs1A10; LBb|f|sPs1A5; LFf|s|bPb5A0; LFf|s|bPs1A0; LFf|s|bPf1A0; LBb|f|sPs1A0; LBb|f|sPb1A0; LFf|s|bPf1A10; LBb|f|sPb5A0; LBb|f|sPb5A10

$F20|\text{prmu}|\sum_{j=1}^{100} U_j$ LFf|s|bPf1A5; LFf|s|bPf1A10; LFf|s|bPf5A0; LFf|s|bPf5A10

$F20|\text{prmu}|\sum_{j=1}^{100} T_j$ LFf|s|bPb1A0; LFf|s|bPb1A5; LFf|s|bPs1A10; LFf|s|bPf1A0; LFf|s|bPb1A10; LBb|f|sPb1A5; LFf|s|bPf1A10; LBb|f|sPb1A0; LFf|s|bPf1A5; LFf|s|bPb10A10; LFf|s|bPs1A0; LFf|s|bPs1A5; LBb|f|sPs1A0; LFf|s|bPb5A5; LBb|f|sPb5A5; LBb|f|sPs1A10; LBb|f|sPf1A5; LBb|f|sPs1A5; LBb|f|sPb10A10; LBb|f|sPf1A0

$F20|\text{prmu}|\sum_{j=1}^{100} w_jU_j$ LFf|s|bPf5A0; LFf|s|bPs10A5; LFf|s|bPs5A5; LFf|s|bPf5A10; LFf|s|bPs1A0; LFf|s|bPf5A5; LFf|s|bPs5A0; LFf|s|bPf1A10; LFf|s|bPb5A0; LBb|f|sPs5A0; LFf|s|bPf10A10; LFf|s|bPf10A5; LFf|s|bPf1A0; LFf|s|bPb5A5; LFf|s|bPs1A5; LFf|s|bPf10A0; LBb|f|sPf10A10; LFf|s|bPs5A10; LFf|s|bPs10A0; LFf|s|bPs10A10; LBb|f|sPs5A5; LFf|s|bPs1A10; LFf|s|bPb5A10; LBb|f|sPf5A10; LBb|f|sPf10A5

$F20|\text{prmu}|\sum_{j=1}^{100} w_jT_j$ LBb|f|sPb1A10; LFf|s|bPf1A10; LFf|s|bPs1A5; LBb|f|sPf1A5; LBb|f|sPf10A10; LFf|s|bPb1A0; LBb|f|sPs1A10; LFf|s|bPb5A5; LBb|f|sPs1A5; LBb|f|sPb1A5; LBb|f|sPb1A0; LFf|s|bPb1A10; LFf|s|bPf1A5; LBb|f|sPb5A5; LFf|s|bPs5A0; LBb|f|sPf1A10; LFf|s|bPs1A0; LFf|s|bPf1A0; LFf|s|bPs1A10; LBb|f|sPb5A0; LBb|f|sPf10A0; LFf|s|bPb5A10; LBb|f|sPf5A5; LFf|s|bPs10A0; LBb|f|sPf5A0; LBb|f|sPs5A0; LFf|s|bPf5A0; LFf|s|bPb5A0; LFf|s|bPf10A0

A.3.2 Main Races

Below is a list of iterated local search candidates selected in the races described in Section 5.2. The names of the candidates reflect the local search they employ and the order of the candidates reflects their score on the ϵ–indicator.

$1||\sum_{j=1}^{100} U_j$ Bb|f|s; Ff|s|b; Ff|b|s; Fs|b|f; Fs|f|; Ff|s|bf; Ff|s|; Fs||; Bf|s|; Bs|b|; Fs|b|; Bs|f|; Fs|bf|; Bb|s|; Bs|bs|f; Ff|bs|; Bs||; Fs|fs|; Bb|s|bf; Fs|f|bf; Ff|s|f; Ff|fs|; Fbs|f|; Fs|f|f; Bs|bsf|

$1||\sum_{j=1}^{100} T_j$ Bb|f|s; Ff|s|b; Bf|b|s; Bf|bf|s; Bb|s|f; Fs|f|b; Bs|f|b; Bf|s|b; Bs|b|f; Fs|b|f; Bb|bf|s; Fs|f|; Ff|s|bf; Bf|b|fs; Ff|b|fs; Bf|fs|b; Bs|fs|b; Bs|b|bf; Bf|s|bf; Fs||; Bs|b|; Fs|b|; Bf|b|bs; Bf|bs|; Bs|f|bs; Ff|s|bs; Bs|b|fs; Bf|b|bsf; Bb|s|fs; Fb|bf|s; Fs|bf|; Fbf|s|; Bbf|s|; Bb|s|; Bs|bf|; Bf|fs|bf; Fs|fs|b; Fs|b|bf; Bf|bs|bf; Ff|b|bs; Fs|f|bsf; Bb|s|bsf; Bf|bsf|; Bfs|b|; Fs|fs|; Bfs|b|s; Bb|s|bf; Bf|bf|bs; Fs|b|fs; Fbf|s|b; Fbf|s|f; Bb|f|fs; Bbs|f|; Bbs|f|b; Bbs|fs|b; Fs|bs|f; Fs|f|bf; Fs|fs|bf; Fs|f|bs; Fs|bs|fs; Fs|b|bsf; Bf|fs|bsf; Fbf|fs|; Fs|f|f; Bs|bsf|

$1||\sum_{j=1}^{100} w_j U_j$ Bf|b|s; Bf|bf|s; Bf|s|b; Bf|b|fs; Bf|s|; Bf|b|bs; Bf|bs|; Bf|fs|bs

$1||\sum_{j=1}^{100} w_j T_j$ Bf|b|s; Bf|bf|s; Bf|s|b; Bf|s|bf; Bf|s|bs; Bf|bf|bs

$P2||\sum_{j=1}^{100} U_j$ Bb|f|s; Bb|s|f; Fs|b|f; Fs|f|; Ff|s|; Fs|b|; Fb|s|f; Fs|bf|; Ffs|b|; Fs|b|bf; Ff|bs|; Bs||; Bb|s|bf

$P2||\sum_{j=1}^{100} T_j$ Ff|b|s; Bf|b|s; Bb|s|f; Fs|f|b; Bs|f|b; Bf|s|b; Bs|b|f; Fs|b|f; Bb|bf|s; Bs|fs|b; Bs|b|; Bf|b|bs; Bf|bs|; Bs|b|bsf; Bb|s|fs; Fs|bf|; Bbf|s|; Fs|fs|b; Bs|f|bf; Bf|bs|bf; Ff|bs|; Fs|f|bsf; Bb|fs|; Bfs|b|s; Bb|s|bf; Bbs|fs|bf; Bbs|f|; Bbs|f|b; Fs|fs|bf; Fs|f|bs; Fs|bs|fs; Bbsf||

$P2||\sum_{j=1}^{100} w_j U_j$ Bb|f|s; Bb|s|fs; Bb|bs|f; Bb|s|bf

$P2||\sum_{j=1}^{100} w_j T_j$ Bb|f|s; Bf|b|s; Bf|bf|s; Bb|s|f; Fs|f|b; Bs|f|b; Bf|s|b; Bs|b|f; Fs|b|f; Bb|bf|s; Bs|fs|b; Bs|b|bf; Bf|s|bf; Bb|f|bs; Bs|b|; Bs|f|; Bf|b|bs; Bf|bs|; Bs|f|bs; Bf|b|bsf; Fs|bf|; Bbf|s|; Bfs|b|f; Bb|bs|f; Bs|bf|; Bf|s|bs; Bf|fs|bs; Bs|f|bsf; Fs|fs|b; Bs|bs|f; Fs|b|bf; Bf|bs|bf; Bs|bs|bf; Bs|bs|fs; Bb|s|bsf; Fs|bsf|; Bbs|bsf|; Bb|fs|; Bbs|bf|; Bf|bsf|; Bfs|b|; Bb|s|bf; Bbs|fs|bf; Bs|bs|; Bf|bf|bs; Fs|b|fs; Bb|f|fs; Bbs|f|; Fs|bs|f; Fs|f|bf; Bbs|f|bf; Fs|f|bs; Fs|b|bsf; Fbs|bf|; Bs|bsf|

$P3||\sum_{j=1}^{100} U_j$ Ff|s|b; Fs|f|b; Fs|b|f; Fs|f|; Ff|s|; Fs||; Fs|b|; Bs|f|; Fs|bf|; Ffs|b|; Bs|f|bf; Ff|bs|; Fs|fs|; Fbs|f|b; Fs|f|bf; Ffs||; Fbs|bf|

$P3||\sum_{j=1}^{100} T_j$ Ff|b|s; Bf|b|s; Fs|f|b; Bs|f|b; Fs|b|f; Fs|bf|; Fs|fs|b; Fs|b|bf; Fs|f|bf; Fs|f|bs

$P3||\sum_{j=1}^{100} w_j U_j$ Bb|f|s; Bf|b|s; Bf|s|b; Bs|b|f; Fs|b|f; Fb|f|s; Bf|b|fs; Bf|fs|b; Bb|f|bs; Bf|bs|; Bs|b|fs; Bbf|s|; Bb|bs|f; Bs|bf|; Bs|f|bf; Bf|bs|bf; Ff|bs|; Bb|fs|; Bf|bsf|

$P3||\sum_{j=1}^{100} w_j T_j$ Bb|f|s; Ff|s|b; Bf|b|s; Fs|f|b; Bs|f|b; Bf|s|b; Bs|b|f; Fs|b|f; Fb|f|s; Ff|s|bf; Bf|b|fs; Bs|fs|b; Bs|b|bf; Bs|b|; Bf|bs|; Bs|f|bs; Ff|s|bs; Bs|b|bsf; Fb|s|f; Fs|bf|; Bbf|s|; Bfs|b|f; Bs|bf|; Bb|bs|bf; Bf|s|bs; Fs|b|bf; Bs|bs|bf; Fs|f|bsf; Fs|bsf|; Bb|fs|; Bfs|b|; Bfs|b|s; Bb|s|bf; Fs|b|fs; Bs|bs|bsf; Fb|s|bf; Bbs|f|; Bbs|f|b; Fs|bs|f; Fs|f|bf; Bbs|f|bf; Fs|fs|bf; Fs|bs|fs; Bbs|fs|; Bs|bsf|

$P4||\sum_{j=1}^{100} U_j$ Ff|s|b; Ff|b|s; Bf|b|s; Bb|s|f; Fs|f|b; Bs|f|b; Bs|b|f; Fs|b|f; Bb|bf|s; Fs|f|; Ff|s|bf; Ff|s|; Ff|bf|s; Bf|fs|b; Bs|b|bf; Bf|s|bf; Fs||; Bf|s|; Fs|b|; Bf|b|bsf; Ff|bsf|; Fb|s|f; Fs|bf|; Ffs|b|; Bs|fs|; Bfs|b|f; Bb|bs|f; Fs|fs|b; Fs|b|bf; Ff|bs|; Fs|bsf|; Fbsf||; Fb|fs|; Fs|fs|; Bb|s|bf; Bf|bf|bs; Fs|b|fs; Fbf|s|b; Fb|bs|f; Fbs|f|b; Bbs|f|; Fs|f|bf; Fs|fs|bf; Fs|bs|fs; Fbs|f|; Ffs||; Fbs|bf|

$P4||\sum_{j=1}^{100} T_j$ Ff|b|s; Bf|b|s; Fs|f|b; Fs|b|f; Ff|bf|s; Bb|f|bs; Fs|b|; Bf|b|bs; Bf|bs|; Ff|s|bs; Fb|s|f; Fb|bf|s; Fs|bf|; Bbf|s|; Fs|b|fs; Fb|bs|f; Fb|s|bf; Fb|fs|bf; Fs|bs|f; Fs|f|bs; Ff|bs|fs; Fs|bs|fs; Fs|b|bsf; Fbs|bf|

$P4||\sum_{j=1}^{100} w_j U_j$ Bb|f|s; Ff|b|s; Bf|b|s; Bf|bf|s; Bb|s|f; Bf|s|b; Bs|b|f; Bb|bf|s; Fb|f|s; Bf|b|fs; Ff|b|fs; Ff|bf|s; Bf|s|bf; Bf|b|bs; Bf|bs|; Fs|bf|; Bb|bs|f; Bb|s|; Bb|bs|bf; Bb|fs|; Bf|bsf|; Bb|s|bf; Bs|bs|; Bbs|f|bf

$P4||\sum_{j=1}^{100} w_j T_j$ Ff|b|s; Bf|b|s; Fs|f|b; Bs|f|b; Bs|b|f; Fs|b|f; Bf|b|fs; Ff|b|fs; Ff|bf|s; Bs|fs|b; Bs|b|bf; Bb|f|bs; Bf|bs|; Bs|f|bs; Bs|b|fs; Bs|b|bsf; Fb|s|f; Fs|bf|; Fbf|s|; Bbf|s|; Bs|bf|; Bs|f|bsf; Fs|fs|b; Bs|bs|f; Fs|b|bf; Bs|f|bf; Ff|bs|; Fs|f|bsf; Bb|fs|; Bbs|bf|; Fbs|fs|; Fs|b|fs; Fbs|f|b; Bbs|f|b; Bs|bs|f; Fs|f|bf; Bbs|f|bf; Fs|f|bs; Fs|bs|fs; Fs|b|bsf; Fbs|f|; Bbs|fs|; Bbsf||

$F20|\text{prmu}|\sum_{j=1}^{50} U_j$ Fs|f|; Ff|s|

$F20|\text{prmu}|\sum_{j=1}^{50} T_j$ Fs|f|b; Bs|b|f; Fbf|s|; Bs|bf|; Fbsf||; Fbf|s|b; Fbf|s|f; Fs|f|bf; Fs|f|bs

$F20|\text{prmu}|\sum_{j=1}^{50} w_j U_j$ Ff|s|

$F20|\text{prmu}|\sum_{j=1}^{50} w_j T_j$ Fs|f|b; Bs|f|b; Bs|b|f; Fs|b|f; Fs|bf|; Fbf|s|; Fs|b|bf; Fbf||; Fbsf||; Fs|b|fs; Fbf|s|b; Fbf|s|f; Fs|f|bf; Fs|f|bs; Fs|b|bsf

$F20|\text{prmu}|\sum_{j=1}^{100} U_j$ Ff|s|

$F20|\text{prmu}|\sum_{j=1}^{100} T_j$ Fs|f|b; Fs|b|f; Fs|bf|; Fbf|s|; Fs|fs|b; Fbf||; Fs|b|fs; Fbf|s|b; Fbf|s|f; Fs|bs|f; Fs|f|bf; Fs|fs|bf; Fs|f|bs; Fs|b|bsf; Fbf|fs|

$F20|\text{prmu}|\sum_{j=1}^{100} w_j U_j$ Ff|s|

$F20|\text{prmu}|\sum_{j=1}^{100} w_j T_j$ Fs|f|b; Fs|b|f; Fs|bf|; Fbf|s|; Fs|fs|b; Fs|b|fs; Fbf|s|b; Fbf|s|f; Fs|bs|f; Fs|f|bf; Fs|fs|bf; Fs|f|bs; Fbf|fs|

B Benchmark Results

Tests were run on a computer with two Intel 2.40GHz processors running Debian Linux 2.24.

Reported are the best known value (bk), deviation from the best known of the solutions optained by the apparent urgency dispatching rule (AU), apparent urgency followed by an insertion heuristic (NEH), and the median deviation from the best known for 10 runs of iterated local search after 1 (ILS.1), 60 (ILS.60), and 300 seconds (ILS.300). The instance are identified by the tree–field representation $\alpha|\beta|\gamma$ at the head of the table, tardiness factor (TF), range of due dates (RDD), and an identification number.

Table B.1 (a): $1||\sum U_j$ (b): $1||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	6	4	2	0.0	0.0	0.0	9	3	3	0.0	0.0	0.0
2	0.2	0.2	6	1	1	0.0	0.0	0.0	11	6	4	1.0	0.0	0.0
3	0.2	0.2	6	2	2	1.0	0.0	0.0	8	8	1	0.0	0.0	0.0
4	0.2	0.2	5	3	3	1.0	0.0	0.0	11	7	2	0.0	0.0	0.0
5	0.2	0.2	6	7	5	1.0	0.0	0.0	10	1	1	0.0	0.0	0.0
6	0.4	0.2	18	11	9	2.0	1.0	0.0	65	26	8	0.0	0.0	0.0
7	0.4	0.2	17	11	10	2.0	0.0	0.0	54	6	1	0.0	0.0	0.0
8	0.4	0.2	18	9	5	2.5	1.0	1.0	53	12	12	0.0	0.0	0.0
9	0.4	0.2	18	9	9	1.0	0.0	0.0	43	31	17	0.0	0.0	0.0
10	0.4	0.2	17	6	5	2.0	0.0	0.0	53	25	2	0.0	0.0	0.0
11	0.6	0.2	29	12	11	2.0	1.0	0.0	127	26	7	1.0	0.0	0.0
12	0.6	0.2	31	14	11	2.0	1.0	1.0	154	26	13	0.5	0.0	0.0
13	0.6	0.2	32	17	15	2.0	1.0	0.0	125	22	12	1.0	0.0	0.0
14	0.6	0.2	30	17	17	4.0	1.0	1.0	116	21	12	1.0	0.0	0.0
15	0.6	0.2	28	12	11	2.5	1.0	1.0	134	25	10	0.0	0.0	0.0
16	0.8	0.2	48	15	12	1.0	0.0	0.0	237	101	68	2.5	0.0	0.0
17	0.8	0.2	46	17	16	2.0	1.0	1.0	195	57	51	1.0	1.0	0.0
18	0.8	0.2	48	16	13	2.0	1.0	0.0	278	101	83	1.5	0.0	0.0
19	0.8	0.2	46	18	16	2.0	1.0	1.0	249	96	85	1.5	1.0	1.0
20	0.8	0.2	44	10	9	1.0	1.0	1.0	227	35	35	2.0	1.0	0.0
21	1.0	0.2	80	13	9	0.0	0.0	0.0	447	76	59	2.0	0.0	0.0
22	1.0	0.2	80	10	10	0.0	0.0	0.0	386	56	40	2.0	2.0	2.0
23	1.0	0.2	78	12	6	1.0	0.0	0.0	390	59	48	1.5	0.0	0.0
24	1.0	0.2	80	13	9	1.0	0.0	0.0	461	77	56	3.0	0.0	0.0
25	1.0	0.2	78	11	9	1.0	0.0	0.0	435	69	53	1.0	0.0	0.0
26	0.2	0.4	1	0	0	0.0	0.0	0.0	1	1	0	0.0	0.0	0.0
27	0.2	0.4	2	4	1	0.0	0.0	0.0	2	11	4	0.0	0.0	0.0
28	0.2	0.4	1	0	0	0.0	0.0	0.0	1	3	1	0.0	0.0	0.0
29	0.2	0.4	1	0	0	0.0	0.0	0.0	2	6	4	0.0	0.0	0.0
30	0.2	0.4	1	0	0	0.0	0.0	0.0	1	2	1	0.0	0.0	0.0
31	0.4	0.4	12	9	9	1.0	1.0	0.0	17	22	10	0.0	0.0	0.0
32	0.4	0.4	12	3	2	0.0	0.0	0.0	23	28	28	2.0	0.0	0.0
33	0.4	0.4	11	13	12	2.0	1.0	1.0	27	31	26	2.0	1.0	0.0
34	0.4	0.4	11	14	12	3.0	1.0	1.0	22	35	18	1.0	0.0	0.0
35	0.4	0.4	10	7	6	0.5	0.0	0.0	25	19	11	0.0	0.0	0.0
36	0.6	0.4	23	9	8	1.0	1.0	1.0	73	37	21	1.5	0.0	0.0
37	0.6	0.4	25	13	11	2.0	2.0	1.0	102	43	21	3.0	1.0	0.5
38	0.6	0.4	24	8	6	1.0	1.0	1.0	61	41	32	3.0	1.0	0.0
39	0.6	0.4	23	13	13	1.0	1.0	0.0	93	54	19	3.0	0.0	0.0
40	0.6	0.4	23	9	5	1.0	0.0	0.0	97	60	31	8.0	1.0	0.0
41	0.8	0.4	44	17	13	2.0	1.0	1.0	193	181	178	9.0	1.5	0.5
42	0.8	0.4	40	17	15	2.0	1.5	1.0	191	153	139	7.0	2.0	1.0
43	0.8	0.4	41	12	9	1.0	1.0	1.0	159	147	140	3.0	0.0	0.0
44	0.8	0.4	37	15	11	3.0	1.5	1.0	175	122	113	3.0	1.0	1.0
45	0.8	0.4	39	18	10	2.0	2.0	1.0	160	119	101	2.0	1.0	1.0
46	1.0	0.4	71	15	12	1.0	0.0	0.0	386	116	104	4.0	0.0	0.0

Table B.1(cntd) (a): $1||\sum U_j$ (b): $1||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
47	1.0	0.4	74	15	13	1.5	0.0	0.0	370	121	95	5.0	1.0	1.0
48	1.0	0.4	69	16	10	2.5	1.0	0.5	404	94	86	6.5	0.5	0.0
49	1.0	0.4	69	14	11	2.0	0.0	0.0	411	114	78	7.0	0.0	0.0
50	1.0	0.4	69	14	13	2.0	1.0	0.0	338	93	93	5.5	0.0	0.0
51	0.2	0.6	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
52	0.2	0.6	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
53	0.2	0.6	0	7	4	0.0	0.0	0.0	0	30	18	0.0	0.0	0.0
54	0.2	0.6	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
55	0.2	0.6	0	3	3	0.0	0.0	0.0	0	12	9	0.0	0.0	0.0
56	0.4	0.6	6	10	9	1.0	0.0	0.0	8	8	7	0.0	0.0	0.0
57	0.4	0.6	6	3	3	1.0	1.0	1.0	12	11	10	0.5	0.0	0.0
58	0.4	0.6	8	9	9	1.0	0.0	0.0	12	9	7	2.0	1.0	1.0
59	0.4	0.6	6	9	4	0.0	0.0	0.0	8	3	2	0.0	0.0	0.0
60	0.4	0.6	7	5	5	0.0	0.0	0.0	15	27	11	1.0	0.0	0.0
61	0.6	0.6	17	12	10	1.0	0.0	0.0	39	17	13	3.0	1.0	1.0
62	0.6	0.6	18	15	13	0.5	0.0	0.0	45	28	17	1.0	0.0	0.0
63	0.6	0.6	17	12	12	0.0	0.0	0.0	51	58	43	7.0	2.0	1.5
64	0.6	0.6	19	13	10	1.5	0.5	0.0	53	45	19	0.0	0.0	0.0
65	0.6	0.6	19	10	9	0.0	0.0	0.0	38	19	17	0.5	0.0	0.0
66	0.8	0.6	35	23	21	5.5	1.0	1.0	135	97	70	13.5	3.0	1.0
67	0.8	0.6	36	19	15	4.0	2.0	2.0	168	93	92	12.0	3.0	1.5
68	0.8	0.6	37	20	17	3.0	1.0	1.0	183	96	90	13.0	1.5	0.0
69	0.8	0.6	34	17	16	2.0	1.0	1.0	140	103	76	7.0	1.0	1.0
70	0.8	0.6	34	17	13	2.0	1.0	0.5	134	59	47	6.0	0.0	0.0
71	1.0	0.6	65	19	13	4.0	0.5	0.0	309	110	106	10.5	0.0	0.0
72	1.0	0.6	62	20	17	4.5	1.5	1.0	304	105	100	18.5	3.0	1.5
73	1.0	0.6	68	20	17	2.0	0.5	0.0	351	133	108	9.5	0.0	0.0
74	1.0	0.6	63	18	13	4.5	1.0	1.0	357	119	75	20.0	1.0	1.0
75	1.0	0.6	63	18	18	3.0	1.0	0.5	345	132	70	12.0	0.0	0.0
76	0.2	0.8	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
77	0.2	0.8	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
78	0.2	0.8	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
79	0.2	0.8	0	2	2	0.0	0.0	0.0	0	6	3	0.0	0.0	0.0
80	0.2	0.8	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
81	0.4	0.8	2	6	5	0.0	0.0	0.0	2	24	19	4.0	0.0	0.0
82	0.4	0.8	2	4	4	0.0	0.0	0.0	2	8	5	1.0	0.0	0.0
83	0.4	0.8	2	4	4	0.0	0.0	0.0	2	10	7	2.0	1.0	0.5
84	0.4	0.8	1	0	0	0.0	0.0	0.0	1	17	8	0.0	0.0	0.0
85	0.4	0.8	1	2	2	0.0	0.0	0.0	1	15	12	1.5	0.0	0.0
86	0.6	0.8	13	15	13	0.0	0.0	0.0	48	53	35	1.5	0.5	0.0
87	0.6	0.8	14	27	26	2.0	1.0	0.0	42	67	51	6.5	2.5	2.0
88	0.6	0.8	14	20	18	0.5	0.0	0.0	31	51	18	2.0	1.0	1.0
89	0.6	0.8	13	21	21	2.0	1.0	1.0	36	50	18	5.5	0.0	0.0
90	0.6	0.8	13	22	20	1.0	0.0	0.0	44	88	57	8.0	3.0	1.0
91	0.8	0.8	37	25	25	5.5	1.0	1.0	174	117	117	6.0	0.0	0.0
92	0.8	0.8	36	20	20	4.0	1.0	1.0	177	107	91	9.0	2.0	2.0
93	0.8	0.8	35	25	17	7.5	1.0	1.0	202	114	92	18.5	3.5	1.0
94	0.8	0.8	35	22	19	4.0	1.0	0.0	186	75	69	9.0	0.0	0.0
95	0.8	0.8	35	24	24	4.0	1.0	1.0	142	102	75	9.5	2.0	0.0
96	1.0	0.8	61	23	22	5.0	1.0	1.0	295	119	106	21.0	2.0	2.0
97	1.0	0.8	59	16	13	4.5	1.0	1.0	333	147	118	15.5	0.5	0.0
98	1.0	0.8	63	21	21	6.0	1.0	1.0	351	109	58	11.0	2.0	1.5
99	1.0	0.8	63	20	15	6.0	2.0	1.0	367	136	122	21.5	1.0	1.0
100	1.0	0.8	53	21	15	6.0	1.0	1.0	266	99	83	15.0	0.5	0.0
101	0.2	1.0	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
102	0.2	1.0	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
103	0.2	1.0	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
104	0.2	1.0	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
105	0.2	1.0	0	0	0	0.0	0.0	0.0	0	0	0	0.0	0.0	0.0
106	0.4	1.0	0	0	0	0.0	0.0	0.0	0	6	3	0.0	0.0	0.0
107	0.4	1.0	2	4	0	0.0	0.0	0.0	2	15	7	3.0	1.0	1.0
108	0.4	1.0	0	0	0	0.0	0.0	0.0	0	13	7	2.0	0.0	0.0
109	0.4	1.0	1	0	0	0.0	0.0	0.0	1	22	11	3.0	0.0	0.0
110	0.4	1.0	0	0	0	0.0	0.0	0.0	0	16	7	2.0	1.0	1.0
111	0.6	1.0	18	29	25	3.0	1.0	1.0	77	58	45	5.0	1.5	0.0
112	0.6	1.0	19	23	23	4.0	1.0	1.0	95	116	106	12.0	4.0	3.0
113	0.6	1.0	13	26	24	2.5	1.0	1.0	53	80	52	6.0	1.0	0.0
114	0.6	1.0	16	30	28	4.0	2.0	1.5	85	99	92	11.0	4.0	2.0
115	0.6	1.0	13	26	24	2.5	1.0	0.0	62	81	65	7.0	0.0	0.0
116	0.8	1.0	39	30	25	8.0	2.0	2.0	192	131	102	28.0	3.0	2.0
117	0.8	1.0	36	27	26	8.0	2.0	1.0	218	112	112	14.5	2.0	1.5
118	0.8	1.0	37	24	16	6.5	2.0	1.0	148	109	85	24.5	4.0	1.0
119	0.8	1.0	36	20	18	6.0	0.5	0.0	181	110	95	16.5	0.0	0.0
120	0.8	1.0	33	20	20	5.5	1.0	0.5	173	75	63	8.5	1.0	1.0
121	1.0	1.0	51	17	16	4.5	1.0	1.0	256	125	115	23.0	3.0	0.0
122	1.0	1.0	56	27	26	7.0	1.0	1.0	317	141	103	15.5	1.0	0.0
123	1.0	1.0	49	20	16	7.0	1.0	0.0	249	109	97	5.5	0.0	0.0
124	1.0	1.0	51	19	18	5.5	0.5	0.0	255	100	71	7.5	0.5	0.0
125	1.0	1.0	50	20	19	5.0	1.0	1.0	259	109	100	18.0	2.0	0.0

Table B.2 (a): $1||\sum T_j$ (b): $1||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	2331	228	214	0	0	0	5988	2658	2573	0	0.0	0
2	0.2	0.2	2198	32	32	32	32	32	6170	1072	992	0	0.0	0
3	0.2	0.2	2356	40	19	19	19	19	4267	3905	3905	0	0.0	0
4	0.2	0.2	1799	337	328	0	0	0	5011	2598	2594	0	0.0	0
5	0.2	0.2	2776	253	250	1	1	1	5283	5216	5216	0	0.0	0
6	0.4	0.2	17859	946	848	0	0	0	58258	4990	4810	0	0.0	0
7	0.4	0.2	16580	541	268	0	0	0	50972	20468	20085	0	0.0	0
8	0.4	0.2	18525	386	347	3	3	3	59434	12987	12514	0	0.0	0
9	0.4	0.2	18248	336	299	0	0	0	40978	5871	5627	0	0.0	0
10	0.4	0.2	16415	668	519	65	65	65	53208	9912	8939	66	0.0	0
11	0.6	0.2	46397	720	533	231	231	231	181649	15686	14566	705	0.0	0
12	0.6	0.2	52906	1978	1746	154	154	154	234179	13183	12173	0	0.0	0
13	0.6	0.2	56493	1867	1643	112	112	112	178840	7171	6455	224	0.0	0
14	0.6	0.2	48231	2497	2150	381	381	381	157476	19324	18308	197	0.5	0
15	0.6	0.2	41990	556	553	79	79	79	172995	20069	18782	0	0.0	0
16	0.8	0.2	100586	589	394	103	103	103	407703	2277	977	0	0.0	0
17	0.8	0.2	95032	1408	1302	0	0	0	332804	7958	6333	508	0.0	0
18	0.8	0.2	102224	1395	920	35	35	35	544838	4236	2618	649	0.0	0
19	0.8	0.2	100060	3082	2734	0	0	0	477684	7848	6510	1538	101.5	0
20	0.8	0.2	89766	1263	738	27	27	27	406094	13701	10327	581	0.0	0
21	1.0	0.2	195778	70	37	0	0	0	898925	145	10	0	0.0	0
22	1.0	0.2	135195	41	11	0	0	0	556873	838	65	0	0.0	0
23	1.0	0.2	135741	82	15	0	0	0	539716	511	59	0	0.0	0
24	1.0	0.2	148978	42	0	0	0	0	744287	410	65	3	0.0	0
25	1.0	0.2	138336	31	16	0	0	0	585306	38	6	0	0.0	0
26	0.2	0.4	1	0	0	0	0	0	8	28	0	0	0.0	0
27	0.2	0.4	337	6	4	0	0	0	718	923	896	0	0.0	0
28	0.2	0.4	3	0	0	0	0	0	27	0	0	0	0.0	0
29	0.2	0.4	201	18	16	0	0	0	480	74	2	0	0.0	0
30	0.2	0.4	10	1	0	0	0	0	50	70	0	0	0.0	0
31	0.4	0.4	11222	346	227	117	117	117	24202	25394	25373	0	0.0	0
32	0.4	0.4	11633	203	26	26	26	26	25469	12871	12871	0	0.0	0
33	0.4	0.4	12946	711	542	104	104	104	32964	16914	16914	0	0.0	0
34	0.4	0.4	11823	866	722	54	54	54	22215	4727	4685	476	0.0	0
35	0.4	0.4	7145	348	232	0	0	0	19114	13491	13031	0	0.0	0
36	0.6	0.4	38654	579	574	51	51	51	108293	26347	26322	688	0.0	0
37	0.6	0.4	50963	1424	1399	1399	91	91	181850	29253	29150	151	0.0	0
38	0.6	0.4	42130	1162	1093	246	246	246	90440	32574	32574	0	0.0	0
39	0.6	0.4	38589	1581	1330	274	274	274	151701	22349	22241	581	0.0	0
40	0.6	0.4	37011	895	875	56	56	56	129728	35238	35238	169	0.0	0
41	0.8	0.4	127870	1226	1211	68	68	68	462324	2097	2010	253	0.0	0
42	0.8	0.4	97797	2466	2466	186	186	186	425875	5462	5462	483	0.0	0
43	0.8	0.4	102279	284	259	15	15	15	320537	5480	5459	103	0.0	0
44	0.8	0.4	83984	1064	1064	10	10	10	360193	1115	1115	49	0.0	0
45	0.8	0.4	91196	254	254	0	0	0	306040	739	739	166	13.0	0
46	1.0	0.4	170293	1	0	0	0	0	829828	247	191	26	0.0	0
47	1.0	0.4	146866	0	0	0	0	0	623356	283	283	1	0.0	0
48	1.0	0.4	140633	21	17	0	0	0	748988	1336	1212	13	0.0	0
49	1.0	0.4	136385	26	26	1	1	1	656693	159	96	17	0.0	0
50	1.0	0.4	144849	5	5	0	0	0	599269	413	226	0	0.0	0
51	0.2	0.6	0	0	0	0	0	0	0	0	0	0	0.0	0
52	0.2	0.6	0	0	0	0	0	0	0	0	0	0	0.0	0
53	0.2	0.6	0	336	336	0	0	0	0	1464	1464	0	0.0	0
54	0.2	0.6	0	0	0	0	0	0	0	0	0	0	0.0	0
55	0.2	0.6	0	78	78	0	0	0	0	471	471	0	0.0	0
56	0.4	0.6	4508	197	163	13	13	13	9046	5987	5985	0	0.0	0
57	0.4	0.6	2850	56	29	0	0	0	11539	8624	8624	0	0.0	0
58	0.4	0.6	7929	356	334	7	7	7	16313	4467	4440	0	0.0	0
59	0.4	0.6	3171	85	54	40	40	40	7965	10639	10623	0	0.0	0
60	0.4	0.6	5850	166	150	0	0	0	19912	10577	10577	3	0.0	0
61	0.6	0.6	41401	1008	934	167	167	167	86793	27182	27182	712	0.0	0
62	0.6	0.6	39294	1177	1170	93	93	93	87067	22014	21926	389	0.0	0
63	0.6	0.6	37105	893	893	113	113	113	96563	14386	14386	13	0.0	0
64	0.6	0.6	35270	1118	1093	597	597	597	100788	35102	35082	395	0.0	0
65	0.6	0.6	27053	393	383	58	58	58	56510	16973	16927	0	0.0	0
66	0.8	0.6	74762	127	125	0	0	0	243872	438	438	176	0.0	0
67	0.8	0.6	89145	234	232	37	37	37	401023	1562	1550	316	0.0	0
68	0.8	0.6	94243	129	129	21	21	21	399085	385	385	53	0.0	0
69	0.8	0.6	85189	97	95	0	0	0	309232	2681	2681	110	0.0	0
70	0.8	0.6	65927	158	158	24	24	24	222684	5045	5045	428	0.0	0
71	1.0	0.6	153836	9	9	0	0	0	640816	912	912	40	0.0	0
72	1.0	0.6	133501	8	8	0	0	0	611362	294	261	67	0.0	0
73	1.0	0.6	141548	13	2	0	0	0	623429	215	38	0	0.0	0
74	1.0	0.6	123681	9	8	0	0	0	584628	722	90	30	0.0	0
75	1.0	0.6	119833	41	37	0	0	0	575274	561	155	9	0.0	0
76	0.2	0.8	0	0	0	0	0	0	0	0	0	0	0.0	0
77	0.2	0.8	0	0	0	0	0	0	0	0	0	0	0.0	0
78	0.2	0.8	0	0	0	0	0	0	0	0	0	0	0.0	0
79	0.2	0.8	0	147	100	0	0	0	0	294	294	0	0.0	0
80	0.2	0.8	0	0	0	0	0	0	0	0	0	0	0.0	0
81	0.4	0.8	468	41	18	0	0	0	1400	853	733	0	0.0	0

Table B.2(cntd) (a): $1||\sum T_j$ (b): $1||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
82	0.4	0.8	266	26	26	0	0	0	317	395	395	0	0.0	0
83	0.4	0.8	489	148	148	0	0	0	1146	507	507	0	0.0	0
84	0.4	0.8	34	0	0	0	0	0	136	0	0	0	0.0	0
85	0.4	0.8	45	22	0	0	0	0	284	197	96	0	0.0	0
86	0.6	0.8	27814	721	721	463	205	205	66850	8890	8850	290	0.0	0
87	0.6	0.8	34961	35	35	0	0	0	84229	8294	8255	74	0.0	0
88	0.6	0.8	26008	265	265	61	61	61	55544	1308	1308	150	0.0	0
89	0.6	0.8	23164	775	775	0	0	0	54612	28189	28189	537	0.0	0
90	0.6	0.8	31722	494	478	290	290	290	75061	5219	5219	449	0.0	0
91	0.8	0.8	77700	47	47	0	0	0	248699	1619	1601	117	0.0	0
92	0.8	0.8	80027	78	78	0	0	0	311022	2170	2101	76	0.0	0
93	0.8	0.8	75098	25	25	0	0	0	326258	1333	1333	61	0.0	0
94	0.8	0.8	71480	197	194	3	3	3	273993	2956	2914	178	0.0	0
95	0.8	0.8	93872	102	102	27	27	27	316870	1766	1758	35	0.0	0
96	1.0	0.8	129247	102	8	0	0	0	495516	802	383	10	0.0	0
97	1.0	0.8	130873	57	11	0	0	0	636903	499	219	6	0.0	0
98	1.0	0.8	157260	92	22	0	0	0	680082	849	331	2	0.0	0
99	1.0	0.8	125053	71	39	5	5	5	622464	28	28	27	0.0	0
100	1.0	0.8	115931	122	96	0	0	0	449545	931	907	70	0.0	0
101	0.2	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
102	0.2	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
103	0.2	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
104	0.2	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
105	0.2	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
106	0.4	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
107	0.4	1.0	437	4	4	0	0	0	1193	273	237	0	0.0	0
108	0.4	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
109	0.4	1.0	29	28	0	0	0	0	232	13	13	0	0.0	0
110	0.4	1.0	0	0	0	0	0	0	0	0	0	0	0.0	0
111	0.6	1.0	49257	208	208	0	0	0	159123	4703	4703	102	0.0	0
112	0.6	1.0	48077	172	170	0	0	0	174367	2653	2645	110	0.0	0
113	0.6	1.0	32582	519	519	0	0	0	91169	3319	3319	191	0.0	0
114	0.6	1.0	47671	129	129	0	0	0	168266	1548	1460	138	0.0	0
115	0.6	1.0	22740	405	400	41	41	41	70190	4397	4397	161	0.0	0
116	0.8	1.0	90020	54	54	0	0	0	370614	872	797	184	0.0	0
117	0.8	1.0	75593	78	73	36	36	36	324437	383	383	0	0.0	0
118	0.8	1.0	80528	71	71	0	0	0	246237	679	649	100	0.0	0
119	0.8	1.0	76720	63	62	0	0	0	293571	999	993	7	0.0	0
120	0.8	1.0	78823	76	76	0	0	0	267316	1637	1629	16	0.0	0
121	1.0	1.0	117229	253	142	0	0	0	471214	1177	375	0	0.0	0
122	1.0	1.0	137139	86	11	0	0	0	570459	453	274	8	0.0	0
123	1.0	1.0	101479	154	71	0	0	0	397029	739	496	0	0.0	0
124	1.0	1.0	121040	143	80	0	0	0	431115	2121	2121	93	0.0	0
125	1.0	1.0	129908	244	163	0	0	0	560754	2079	1384	332	0.0	0

Table B.3 (a): $P2||\sum U_j$ (b): $P2||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	6	5	1	0.0	0.0	0.0	9	24	2	0.0	0.0	0.0
2	0.2	0.2	6	5	0	0.0	0.0	0.0	12	20	2	0.0	0.0	0.0
3	0.2	0.2	6	6	1	1.0	1.0	0.5	8	18	1	0.0	0.0	0.0
4	0.2	0.2	6	4	0	0.0	0.0	0.0	11	27	3	0.0	0.0	0.0
5	0.2	0.2	6	4	3	0.0	0.0	0.0	10	26	2	0.0	0.0	0.0
6	0.4	0.2	18	6	1	0.0	0.0	0.0	66	27	1	0.0	0.0	0.0
7	0.4	0.2	17	6	0	0.0	0.0	0.0	55	28	8	1.0	0.0	0.0
8	0.4	0.2	18	5	1	1.0	0.0	0.0	53	24	3	0.0	0.0	0.0
9	0.4	0.2	18	6	0	0.0	0.0	0.0	43	38	2	1.0	0.0	0.0
10	0.4	0.2	17	5	1	0.0	0.0	0.0	54	24	6	0.0	0.0	0.0
11	0.6	0.2	29	8	1	0.0	0.0	0.0	129	41	2	1.0	0.0	0.0
12	0.6	0.2	31	6	1	1.0	0.0	0.0	154	46	9	2.0	0.0	0.0
13	0.6	0.2	32	12	1	1.0	0.0	0.0	126	49	5	1.5	0.0	0.0
14	0.6	0.2	30	7	1	1.0	0.0	0.0	117	31	3	1.0	0.0	0.0
15	0.6	0.2	29	8	1	0.0	0.0	0.0	135	25	4	1.5	0.5	0.0
16	0.8	0.2	48	8	0	0.0	0.0	0.0	240	59	14	2.5	1.0	0.0
17	0.8	0.2	46	13	1	1.0	0.0	0.0	196	58	5	2.0	0.0	0.0
18	0.8	0.2	48	11	0	0.0	0.0	0.0	280	53	1	1.0	0.0	0.0
19	0.8	0.2	47	10	2	1.0	1.0	0.0	250	92	9	5.0	2.0	1.0
20	0.8	0.2	45	6	0	0.0	0.0	0.0	228	60	4	2.0	0.0	0.0
21	1.0	0.2	80	10	3	1.0	1.0	1.0	450	65	28	1.0	1.0	0.0
22	1.0	0.2	80	10	4	0.0	0.0	0.0	391	51	25	3.0	0.0	0.0
23	1.0	0.2	79	10	4	1.5	1.0	1.0	397	52	17	4.5	1.0	0.0
24	1.0	0.2	81	10	3	1.0	0.0	0.0	469	70	35	3.0	0.0	0.0
25	1.0	0.2	79	9	3	0.5	0.0	0.0	436	78	26	2.0	0.0	0.0
26	0.2	0.4	1	5	1	0.0	0.0	0.0	1	16	6	0.0	0.0	0.0
27	0.2	0.4	2	7	1	1.0	0.0	0.0	2	22	1	0.0	0.0	0.0
28	0.2	0.4	1	3	1	0.0	0.0	0.0	1	20	5	0.0	0.0	0.0
29	0.2	0.4	2	6	1	0.0	0.0	0.0	2	28	4	0.0	0.0	0.0
30	0.2	0.4	1	7	2	0.0	0.0	0.0	1	21	6	0.0	0.0	0.0

Table B.3(cntd) (a): $P2||\sum U_j$ (b): $P2||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
31	0.4	0.4	12	10	1	1.0	0.0	0.0	17	25	5	1.0	0.0	0.0
32	0.4	0.4	12	11	1	1.0	0.0	0.0	24	50	8	1.5	0.0	0.0
33	0.4	0.4	12	11	1	0.0	0.0	0.0	27	48	5	2.0	2.0	1.5
34	0.4	0.4	11	7	1	0.0	0.0	0.0	23	46	3	2.0	0.5	0.0
35	0.4	0.4	10	9	1	0.0	0.0	0.0	26	32	3	0.0	0.0	0.0
36	0.6	0.4	24	13	0	0.0	0.0	0.0	73	39	14	2.0	1.0	0.0
37	0.6	0.4	26	14	1	1.0	0.0	0.0	103	82	4	2.5	0.0	0.0
38	0.6	0.4	24	11	1	1.0	0.0	0.0	62	69	9	2.0	0.0	0.0
39	0.6	0.4	23	14	2	1.0	0.0	0.0	94	78	11	3.0	0.0	0.0
40	0.6	0.4	23	9	1	0.0	0.0	0.0	98	77	14	6.5	1.0	0.0
41	0.8	0.4	45	19	2	1.0	0.0	0.0	197	118	34	10.0	2.5	2.0
42	0.8	0.4	41	18	5	2.0	1.0	1.0	193	127	29	8.0	3.5	2.0
43	0.8	0.4	42	11	1	1.0	0.0	0.0	159	82	37	8.0	3.0	2.0
44	0.8	0.4	38	14	3	2.0	1.0	1.0	177	75	12	5.0	0.0	0.0
45	0.8	0.4	41	14	3	0.0	0.0	0.0	167	93	31	2.0	0.0	0.0
46	1.0	0.4	72	15	5	1.0	0.0	0.0	391	111	33	4.0	0.0	0.0
47	1.0	0.4	74	17	7	2.0	0.5	0.0	370	112	52	8.0	3.0	2.5
48	1.0	0.4	69	15	6	3.0	1.0	1.0	406	93	39	9.0	0.0	0.0
49	1.0	0.4	70	15	7	2.0	0.0	0.0	415	95	36	3.0	0.0	0.0
50	1.0	0.4	70	14	5	1.0	0.0	0.0	344	79	34	4.0	0.0	0.0
51	0.2	0.6	0	12	1	0.0	0.0	0.0	0	30	3	0.0	0.0	0.0
52	0.2	0.6	0	10	0	0.0	0.0	0.0	0	28	2	0.0	0.0	0.0
53	0.2	0.6	0	11	1	0.0	0.0	0.0	0	55	5	0.0	0.0	0.0
54	0.2	0.6	0	10	2	0.0	0.0	0.0	0	21	0	0.0	0.0	0.0
55	0.2	0.6	0	6	4	0.0	0.0	0.0	0	32	2	0.0	0.0	0.0
56	0.4	0.6	7	14	2	0.0	0.0	0.0	9	59	10	1.0	0.0	0.0
57	0.4	0.6	7	10	2	1.0	0.0	0.0	12	53	25	1.5	0.0	0.0
58	0.4	0.6	9	17	1	1.0	1.0	0.0	13	61	11	3.0	1.0	0.0
59	0.4	0.6	6	16	2	1.5	0.0	0.0	8	55	22	1.0	0.0	0.0
60	0.4	0.6	7	14	1	1.0	0.0	0.0	15	57	13	4.0	1.0	0.0
61	0.6	0.6	18	15	3	1.0	0.0	0.0	41	81	20	6.0	2.0	1.0
62	0.6	0.6	18	17	3	1.0	1.0	0.0	46	100	16	6.5	2.0	1.5
63	0.6	0.6	17	14	1	1.0	0.0	0.0	53	90	21	11.5	3.0	2.0
64	0.6	0.6	19	13	1	1.0	0.0	0.0	54	64	5	4.0	0.0	0.0
65	0.6	0.6	19	14	1	1.0	0.0	0.0	38	59	13	1.5	0.0	0.0
66	0.8	0.6	36	23	8	3.0	1.0	1.0	140	98	23	8.0	3.0	0.0
67	0.8	0.6	39	19	7	2.0	1.0	0.0	181	116	45	11.5	3.0	1.0
68	0.8	0.6	38	23	6	2.0	1.0	0.0	186	127	57	10.0	4.0	3.0
69	0.8	0.6	35	17	4	1.5	0.0	0.0	145	122	26	10.0	2.5	1.0
70	0.8	0.6	34	19	6	3.0	1.0	0.5	134	110	27	8.5	3.0	2.0
71	1.0	0.6	66	18	7	3.5	0.0	0.0	310	109	60	12.0	3.0	0.0
72	1.0	0.6	63	20	9	3.0	1.0	1.0	310	109	52	9.5	2.0	1.0
73	1.0	0.6	68	20	8	3.0	1.0	1.0	354	130	41	7.0	0.0	0.0
74	1.0	0.6	65	16	5	2.5	0.0	0.0	365	104	37	11.5	0.0	0.0
75	1.0	0.6	63	18	7	4.0	1.0	1.0	345	132	31	9.0	3.0	3.0
76	0.2	0.8	0	10	1	0.0	0.0	0.0	0	28	2	0.0	0.0	0.0
77	0.2	0.8	0	4	0	0.0	0.0	0.0	0	11	0	0.0	0.0	0.0
78	0.2	0.8	0	11	2	0.0	0.0	0.0	0	38	1	0.0	0.0	0.0
79	0.2	0.8	0	9	1	0.0	0.0	0.0	0	32	1	0.0	0.0	0.0
80	0.2	0.8	0	9	3	0.0	0.0	0.0	0	35	11	0.0	0.0	0.0
81	0.4	0.8	2	15	5	4.0	1.0	1.0	3	79	15	6.5	3.0	2.0
82	0.4	0.8	3	14	5	3.5	0.0	0.0	2	67	4	2.0	1.0	1.0
83	0.4	0.8	3	18	5	2.0	1.0	0.0	2	65	13	3.0	1.0	1.0
84	0.4	0.8	1	16	4	1.0	0.0	0.0	1	66	9	1.0	0.0	0.0
85	0.4	0.8	1	15	7	4.0	2.0	1.0	2	35	12	3.0	0.5	0.0
86	0.6	0.8	14	19	4	3.0	0.5	0.0	54	95	25	4.5	0.5	0.0
87	0.6	0.8	16	17	6	3.0	0.0	0.0	46	121	28	14.5	3.0	2.0
88	0.6	0.8	14	24	5	1.0	0.0	0.0	37	83	16	4.0	0.0	0.0
89	0.6	0.8	15	20	6	2.0	1.0	0.0	43	102	24	8.5	1.5	1.0
90	0.6	0.8	14	18	4	2.0	0.0	0.0	47	112	28	9.0	3.5	2.0
91	0.8	0.8	38	24	11	4.5	1.0	1.0	174	139	33	7.0	0.0	0.0
92	0.8	0.8	37	23	10	4.5	1.0	1.0	179	123	34	8.5	2.0	0.5
93	0.8	0.8	36	25	8	3.0	1.0	1.0	206	118	40	13.0	2.0	2.0
94	0.8	0.8	36	23	8	2.0	0.0	0.0	191	91	36	7.0	0.0	0.0
95	0.8	0.8	36	26	9	3.0	1.0	0.0	146	107	49	13.0	3.0	2.0
96	1.0	0.8	61	23	12	4.0	1.0	1.0	297	113	50	14.0	4.0	1.0
97	1.0	0.8	60	15	8	3.0	0.0	0.0	339	131	56	8.0	0.0	0.0
98	1.0	0.8	64	21	7	4.5	1.0	0.0	356	104	29	6.0	0.0	0.0
99	1.0	0.8	64	19	10	5.0	2.0	1.0	370	148	68	17.0	1.0	1.0
100	1.0	0.8	54	20	8	4.0	1.0	0.0	271	104	39	13.0	1.0	0.5
101	0.2	1.0	0	7	1	0.0	0.0	0.0	0	22	0	0.0	0.0	0.0
102	0.2	1.0	0	2	0	0.0	0.0	0.0	0	10	0	0.0	0.0	0.0
103	0.2	1.0	0	8	0	0.0	0.0	0.0	0	25	0	0.0	0.0	0.0
104	0.2	1.0	0	12	0	0.0	0.0	0.0	0	27	0	0.0	0.0	0.0
105	0.2	1.0	0	5	0	0.0	0.0	0.0	0	15	2	0.0	0.0	0.0
106	0.4	1.0	0	13	2	2.0	0.0	0.0	0	25	4	0.0	0.0	0.0
107	0.4	1.0	3	20	4	3.0	1.0	0.0	3	48	15	5.0	1.0	0.0
108	0.4	1.0	1	15	4	3.0	1.0	0.0	0	51	17	3.0	1.0	1.0
109	0.4	1.0	2	16	5	4.0	1.0	0.5	2	53	17	8.0	1.5	1.0
110	0.4	1.0	1	14	5	4.0	1.0	1.0	1	61	16	5.0	1.0	1.0
111	0.6	1.0	20	22	7	2.0	0.0	0.0	80	80	22	6.5	1.0	0.0

Table B.3(cntd) (a): $P2||\sum U_j$ (b): $P2||\sum w_jU_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
112	0.6	1.0	21	22	7	2.0	1.0	0.0	100	120	39	11.5	3.0	3.0
113	0.6	1.0	14	21	7	4.0	1.0	1.0	55	103	31	10.5	2.0	2.0
114	0.6	1.0	18	25	9	4.0	1.0	1.0	89	118	46	11.0	3.5	1.0
115	0.6	1.0	14	24	7	3.0	1.0	0.0	67	119	26	6.5	0.0	0.0
116	0.8	1.0	41	27	10	5.0	0.5	0.0	200	132	42	17.0	4.0	3.0
117	0.8	1.0	37	27	11	5.0	1.0	0.0	219	129	35	10.5	1.0	1.0
118	0.8	1.0	38	24	9	5.0	1.5	1.0	159	98	33	14.5	3.0	1.0
119	0.8	1.0	37	21	6	5.0	0.0	0.0	181	119	39	17.0	0.0	0.0
120	0.8	1.0	34	21	9	5.0	1.0	1.0	178	72	16	6.0	1.0	1.0
121	1.0	1.0	52	16	9	4.0	1.0	0.0	261	124	43	13.0	1.5	0.5
122	1.0	1.0	56	27	12	6.0	2.0	1.0	317	161	52	4.5	1.0	1.0
123	1.0	1.0	49	21	10	5.0	1.0	1.0	249	108	38	6.0	3.0	1.5
124	1.0	1.0	52	20	9	4.0	0.0	0.0	256	99	31	4.5	0.0	0.0
125	1.0	1.0	52	19	8	3.0	0.0	0.0	267	115	45	12.5	2.0	1.0

Table B.4 (a): $P2||\sum T_j$ (b): $P2||\sum w_jT_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	1320	1138	725	0.0	0.0	0.0	3339	2186	589	1e+00	0.0	0.0
2	0.2	0.2	1255	986	621	0.0	0.0	0.0	3461	2008	756	1e+00	0.0	0.0
3	0.2	0.2	1334	1058	807	0.0	0.0	0.0	2359	2582	1123	0e+00	0.0	0.0
4	0.2	0.2	1036	909	473	0.0	0.0	0.0	2786	3663	1255	0e+00	0.0	0.0
5	0.2	0.2	1550	1307	1023	0.0	0.0	0.0	2903	2342	951	0e+00	0.0	0.0
6	0.4	0.2	9358	1817	1203	1.0	0.0	0.0	30665	8083	4316	1e+01	0.0	0.0
7	0.4	0.2	8697	2665	2011	65.0	0.0	0.0	26809	10207	6022	2e+00	0.0	0.0
8	0.4	0.2	9718	2103	1098	0.0	0.0	0.0	31099	7166	4251	2e+00	0.0	0.0
9	0.4	0.2	9563	3353	2415	12.0	0.0	0.0	21451	8597	3390	5e-01	0.0	0.0
10	0.4	0.2	8635	1523	789	8.0	0.0	0.0	27845	6258	2845	2e+01	0.0	0.0
11	0.6	0.2	23907	2575	1640	44.0	1.0	1.0	93894	13863	7875	3e+02	3.5	0.0
12	0.6	0.2	27184	2886	1874	33.0	0.0	0.0	120561	16699	9797	2e+02	58.5	4.0
13	0.6	0.2	29013	3205	1585	15.5	4.0	0.0	92567	15833	8647	1e+02	9.5	5.5
14	0.6	0.2	24803	1993	1361	4.0	0.0	0.0	81386	8767	4719	6e+01	3.5	0.0
15	0.6	0.2	21633	2350	1479	10.0	2.0	0.0	89499	10333	4887	3e+01	3.5	0.0
16	0.8	0.2	51326	2700	1300	75.5	5.0	5.0	209109	19254	9163	2e+02	18.5	7.0
17	0.8	0.2	48512	2213	765	64.0	0.0	0.0	171029	14770	7417	4e+02	76.5	42.0
18	0.8	0.2	52136	2299	667	72.5	0.0	0.0	278686	13766	6431	5e+02	18.5	9.5
19	0.8	0.2	51085	2874	1411	23.0	2.0	1.0	244785	12186	3977	2e+02	70.5	48.5
20	0.8	0.2	45889	3272	1345	43.0	4.0	0.0	208208	17680	7571	5e+02	146.5	18.0
21	1.0	0.2	99322	397	57	2.0	0.0	0.0	457837	2368	510	5e+01	23.0	9.0
22	1.0	0.2	68704	204	9	2.0	0.0	0.0	284132	1601	169	5e+01	13.0	5.5
23	1.0	0.2	69005	241	18	2.0	0.0	0.0	275649	1439	343	6e+01	15.5	8.5
24	1.0	0.2	75687	249	26	1.5	0.0	0.0	379318	1372	226	4e+01	13.5	8.5
25	1.0	0.2	70346	263	33	6.0	0.0	0.0	299063	1186	177	3e+01	8.0	4.5
26	0.2	0.4	22	1331	513	0.0	0.0	0.0	92	3767	1902	0e+00	0.0	0.0
27	0.2	0.4	251	1712	596	0.5	0.0	0.0	504	3029	67	3e+00	0.0	0.0
28	0.2	0.4	14	802	254	0.0	0.0	0.0	84	4668	1061	0e+00	0.0	0.0
29	0.2	0.4	155	2080	665	0.0	0.0	0.0	390	5896	832	0e+00	0.0	0.0
30	0.2	0.4	11	2337	961	0.0	0.0	0.0	55	4780	372	0e+00	0.0	0.0
31	0.4	0.4	5992	4182	1806	4.5	0.0	0.0	12729	15036	7483	0e+00	0.0	0.0
32	0.4	0.4	6200	5409	2457	13.5	0.0	0.0	13641	23525	10705	2e+02	0.0	0.0
33	0.4	0.4	6900	4859	2431	59.5	0.0	0.0	17557	20967	7891	1e+02	2.0	2.0
34	0.4	0.4	6283	3709	1706	4.5	0.0	0.0	12100	12560	4942	4e+01	0.0	0.0
35	0.4	0.4	3850	3364	1382	38.0	0.0	0.0	10442	10464	4830	7e+01	0.5	0.0
36	0.6	0.4	20014	5935	1658	71.0	0.0	0.0	56671	20583	8273	5e+01	1.0	0.5
37	0.6	0.4	26313	6315	1686	58.5	1.0	1.0	94179	48037	23251	7e+02	20.5	14.5
38	0.6	0.4	21780	4859	2225	29.5	0.0	0.0	47361	32703	12943	3e+02	27.0	27.0
39	0.6	0.4	20001	6317	2384	101.5	6.0	1.5	78790	31016	12244	6e+02	186.0	1.0
40	0.6	0.4	19124	5708	2904	138.5	10.5	4.5	67302	38549	18599	2e+02	36.0	20.5
41	0.8	0.4	65219	4121	475	64.5	1.0	0.0	238032	26862	3703	4e+02	142.5	70.5
42	0.8	0.4	49987	4139	854	64.0	9.0	1.5	218492	20628	2474	1e+03	113.0	62.0
43	0.8	0.4	52317	3724	768	123.0	1.0	0.0	165336	17473	4609	9e+02	48.5	30.0
44	0.8	0.4	43060	4278	663	65.0	3.0	2.0	185623	20646	4897	5e+02	80.0	47.5
45	0.8	0.4	46757	4206	544	72.0	11.5	3.5	158342	26629	3629	6e+02	143.0	83.0
46	1.0	0.4	86499	903	61	10.5	0.0	0.0	422846	5139	418	9e+01	30.5	9.0
47	1.0	0.4	74663	549	40	0.0	0.0	0.0	318173	3888	324	7e+01	19.5	11.5
48	1.0	0.4	71536	787	123	4.0	0.0	0.0	382153	5735	497	1e+02	39.5	26.0
49	1.0	0.4	69391	767	64	6.0	0.0	0.0	335297	3871	848	6e+01	21.5	10.0
50	1.0	0.4	73641	617	120	5.5	0.0	0.0	305918	5533	521	1e+02	33.0	11.5
51	0.2	0.6	0	3541	36	0.0	0.0	0.0	0	11540	1036	0e+00	0.0	0.0
52	0.2	0.6	0	1821	0	0.0	0.0	0.0	0	4934	87	0e+00	0.0	0.0
53	0.2	0.6	0	3750	117	0.0	0.0	0.0	0	17667	475	0e+00	0.0	0.0
54	0.2	0.6	0	3662	233	0.0	0.0	0.0	0	6978	0	0e+00	0.0	0.0
55	0.2	0.6	0	4795	809	0.0	0.0	0.0	0	12430	179	0e+00	0.0	0.0
56	0.4	0.6	2568	7672	3544	6.5	0.0	0.0	5047	18951	5341	1e+02	95.0	71.0
57	0.4	0.6	1613	10168	4410	7.0	0.0	0.0	6376	35358	12220	5e+01	0.0	0.0
58	0.4	0.6	4349	9207	2430	46.5	4.0	0.0	8986	25789	8791	2e+02	38.0	3.5
59	0.4	0.6	1769	7309	2389	0.0	0.0	0.0	4320	24535	6246	0e+00	0.0	0.0
60	0.4	0.6	3209	8328	2445	0.0	0.0	0.0	10646	34591	12900	3e+01	0.0	0.0

Table B.4(cntd) (a): $P2||\sum T_j$ (b): $P2||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
61	0.6	0.6	21429	9382	2766	199.0	28.5	6.5	45610	40013	9545	8e+02	72.5	32.0
62	0.6	0.6	20368	11070	3313	155.0	15.5	7.5	45673	55815	17514	8e+02	68.5	28.5
63	0.6	0.6	19323	8712	2555	220.0	11.5	3.5	51833	55247	10154	9e+02	180.0	149.5
64	0.6	0.6	18343	9824	4203	81.0	1.0	0.0	52736	31907	8848	2e+02	0.0	0.0
65	0.6	0.6	14101	9799	3181	123.0	3.5	1.0	29374	33041	11510	7e+01	0.0	0.0
66	0.8	0.6	38449	4557	712	58.0	8.0	2.0	126559	25591	4914	4e+02	83.5	30.5
67	0.8	0.6	45701	5582	877	85.5	7.0	2.0	206665	34009	3989	7e+02	216.0	110.0
68	0.8	0.6	48353	6143	573	58.5	11.0	3.5	206119	34962	3926	4e+02	81.5	21.0
69	0.8	0.6	43717	5112	515	88.0	1.0	0.5	160431	30870	2391	5e+02	136.5	102.5
70	0.8	0.6	33990	5728	914	112.0	16.5	7.0	115494	28519	6904	9e+02	120.5	49.5
71	1.0	0.6	78205	1535	82	15.5	0.5	0.0	327319	8596	1091	1e+02	35.0	24.5
72	1.0	0.6	68015	1608	101	9.0	2.0	1.0	312493	10784	945	2e+02	33.5	17.5
73	1.0	0.6	72042	1240	98	11.5	0.0	0.0	318861	4261	331	4e+01	10.5	4.5
74	1.0	0.6	62968	2064	283	14.5	0.0	0.0	298665	8706	1099	1e+02	47.0	16.5
75	1.0	0.6	61071	1330	115	9.0	1.0	1.0	294344	4883	290	6e+01	18.0	13.5
76	0.2	0.8	0	4720	96	0.0	0.0	0.0	0	12867	234	0e+00	0.0	0.0
77	0.2	0.8	0	1039	0	0.0	0.0	0.0	0	2848	0	0e+00	0.0	0.0
78	0.2	0.8	0	7347	137	0.0	0.0	0.0	0	16613	73	0e+00	0.0	0.0
79	0.2	0.8	0	5503	71	0.0	0.0	0.0	0	15630	186	0e+00	0.0	0.0
80	0.2	0.8	0	5884	533	0.0	0.0	0.0	0	19245	1495	0e+00	0.0	0.0
81	0.4	0.8	365	11152	2552	30.0	0.0	0.0	908	41413	8661	3e+02	1.0	0.0
82	0.4	0.8	248	11970	3059	22.0	1.0	1.0	354	23213	4577	3e+01	10.5	0.0
83	0.4	0.8	310	12804	3730	10.0	0.0	0.0	698	40888	7904	5e+02	0.0	0.0
84	0.4	0.8	48	7705	1601	0.0	0.0	0.0	192	28005	5176	0e+00	0.0	0.0
85	0.4	0.8	94	13604	4128	2.0	0.0	0.0	307	32943	9052	5e+01	16.0	8.0
86	0.6	0.8	14778	14549	4274	255.5	33.5	16.5	36697	55307	13654	1e+03	189.5	59.0
87	0.6	0.8	18542	13487	2570	170.0	25.0	10.5	46027	66887	15177	8e+02	222.5	88.5
88	0.6	0.8	13960	11800	980	139.0	23.0	7.0	30390	51980	10372	5e+02	82.5	28.0
89	0.6	0.8	12396	9904	2392	103.0	6.5	2.0	29310	34836	8990	6e+02	68.0	13.0
90	0.6	0.8	16795	12560	2485	273.0	25.5	9.0	41794	58708	7971	7e+02	156.0	92.0
91	0.8	0.8	39950	6385	431	94.5	11.5	4.0	129993	27345	5703	4e+02	90.5	44.0
92	0.8	0.8	41164	7264	662	102.0	8.0	1.5	161341	40745	6050	5e+02	79.5	53.5
93	0.8	0.8	38563	5317	393	42.0	12.5	7.0	168711	33822	3849	3e+02	59.5	26.5
94	0.8	0.8	36839	6295	537	68.5	5.0	2.0	142292	36106	4875	5e+02	144.5	45.0
95	0.8	0.8	48213	7022	398	64.0	11.0	3.5	164499	47738	5290	6e+02	170.0	86.0
96	1.0	0.8	65846	1289	165	10.5	0.0	0.0	254222	11852	1113	2e+02	51.0	18.0
97	1.0	0.8	66655	2164	261	29.5	1.0	1.0	325878	8506	342	1e+02	35.5	25.5
98	1.0	0.8	79945	1478	63	4.5	0.0	0.0	347063	6224	317	2e+02	63.5	16.5
99	1.0	0.8	63711	2334	189	18.0	1.0	0.5	318542	10678	862	8e+01	20.0	6.5
100	1.0	0.8	59115	3048	253	14.5	2.0	0.0	230420	14587	1074	3e+02	73.5	30.5
101	0.2	1.0	0	3179	27	0.0	0.0	0.0	0	9218	0	0e+00	0.0	0.0
102	0.2	1.0	0	2400	0	0.0	0.0	0.0	0	6001	0	0e+00	0.0	0.0
103	0.2	1.0	0	2794	0	0.0	0.0	0.0	0	10484	0	0e+00	0.0	0.0
104	0.2	1.0	0	4397	0	0.0	0.0	0.0	0	12138	0	0e+00	0.0	0.0
105	0.2	1.0	0	2281	0	0.0	0.0	0.0	0	8375	18	0e+00	0.0	0.0
106	0.4	1.0	0	7917	890	0.0	0.0	0.0	0	13076	1527	0e+00	0.0	0.0
107	0.4	1.0	344	18668	2290	2.5	0.0	0.0	774	46572	12009	6e+01	0.0	0.0
108	0.4	1.0	0	13452	1937	0.0	0.0	0.0	0	35936	6170	0e+00	0.0	0.0
109	0.4	1.0	36	14174	2692	0.0	0.0	0.0	204	39685	8950	6e+01	0.0	0.0
110	0.4	1.0	0	13514	2151	0.0	0.0	0.0	0	55620	6810	0e+00	0.0	0.0
111	0.6	1.0	25792	13525	1615	136.0	30.0	10.0	84380	83095	11726	8e+02	222.5	74.0
112	0.6	1.0	25122	12168	1172	145.5	20.0	7.0	92130	61507	5763	1e+03	244.5	135.0
113	0.6	1.0	17333	17145	1000	228.5	24.5	8.0	49285	74685	10803	7e+02	219.5	88.5
114	0.6	1.0	25039	13665	632	89.0	29.5	4.0	89466	64261	11387	7e+02	238.0	78.5
115	0.6	1.0	12280	13428	1811	208.0	29.5	13.5	38751	61816	6077	1e+03	139.0	40.0
116	0.8	1.0	46187	5692	312	42.5	13.5	1.0	191263	39252	1989	4e+02	120.0	66.0
117	0.8	1.0	38931	7131	373	94.0	6.5	2.5	168559	47039	3180	7e+02	224.0	101.0
118	0.8	1.0	41358	7975	509	119.0	8.0	3.0	127954	42061	4210	4e+02	81.0	29.0
119	0.8	1.0	39431	7329	351	86.0	17.0	4.0	152768	41762	3582	5e+02	193.5	99.5
120	0.8	1.0	40464	8160	655	104.5	28.0	6.0	138422	40118	2396	3e+02	110.0	46.0
121	1.0	1.0	59815	4671	437	52.5	2.0	0.0	242042	27335	1422	3e+02	91.0	53.0
122	1.0	1.0	69849	2333	69	4.5	1.0	0.0	292312	13225	504	2e+02	47.5	31.0
123	1.0	1.0	51875	4584	374	38.0	4.0	4.0	204341	19987	1617	4e+02	77.5	43.0
124	1.0	1.0	61735	4123	286	58.0	2.5	1.5	221705	27448	1493	2e+02	90.0	46.0
125	1.0	1.0	66205	4354	532	37.5	3.0	2.0	287279	32127	1916	4e+02	119.5	43.0

Table B.5 (a): $P3||\sum U_j$ (b): $P3||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	6	5	1	0.0	0.0	0.0	9	24	2	0.0	0.0	0.0
2	0.2	0.2	6	5	0	0.0	0.0	0.0	12	20	2	0.0	0.0	0.0
3	0.2	0.2	6	6	1	1.0	1.0	0.5	8	18	1	0.0	0.0	0.0
4	0.2	0.2	6	4	0	0.0	0.0	0.0	11	27	3	0.0	0.0	0.0
5	0.2	0.2	6	4	3	0.0	0.0	0.0	10	26	2	0.0	0.0	0.0
6	0.4	0.2	18	6	1	0.0	0.0	0.0	66	27	1	0.0	0.0	0.0
7	0.4	0.2	17	6	0	0.0	0.0	0.0	55	28	8	1.0	0.0	0.0
8	0.4	0.2	18	5	1	1.0	0.0	0.0	53	24	3	0.0	0.0	0.0
9	0.4	0.2	18	6	0	0.0	0.0	0.0	43	38	2	1.0	0.0	0.0

Table B.5(cntd) (a): $P3||\sum U_j$ (b): $P3||\sum w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
10	0.4	0.2	17	5	1	0.0	0.0	0.0	54	24	6	0.0	0.0	0.0
11	0.6	0.2	29	8	1	0.0	0.0	0.0	129	41	2	1.0	0.0	0.0
12	0.6	0.2	31	6	1	1.0	0.0	0.0	154	46	9	2.0	0.0	0.0
13	0.6	0.2	32	12	1	1.0	0.0	0.0	126	49	5	1.5	0.0	0.0
14	0.6	0.2	30	7	1	1.0	0.0	0.0	117	31	3	1.0	0.0	0.0
15	0.6	0.2	29	8	1	0.0	0.0	0.0	135	25	4	1.5	0.5	0.0
16	0.8	0.2	48	8	0	0.0	0.0	0.0	240	59	14	2.5	1.0	0.0
17	0.8	0.2	46	13	1	1.0	0.0	0.0	196	58	5	2.0	0.0	0.0
18	0.8	0.2	48	11	0	0.0	0.0	0.0	280	53	1	1.0	0.0	0.0
19	0.8	0.2	47	10	2	1.0	1.0	0.0	250	92	9	5.0	2.0	1.0
20	0.8	0.2	45	6	0	0.0	0.0	0.0	228	60	4	2.0	0.0	0.0
21	1.0	0.2	80	10	3	1.0	1.0	1.0	450	65	28	1.0	1.0	0.0
22	1.0	0.2	80	10	4	0.0	0.0	0.0	391	51	25	3.0	0.0	0.0
23	1.0	0.2	79	10	4	1.5	1.0	1.0	397	52	17	4.5	1.0	0.0
24	1.0	0.2	81	10	3	1.0	0.0	0.0	469	70	35	3.0	0.0	0.0
25	1.0	0.2	79	9	3	0.5	0.0	0.0	436	78	26	2.0	0.0	0.0
26	0.2	0.4	1	5	1	0.0	0.0	0.0	1	16	6	0.0	0.0	0.0
27	0.2	0.4	2	7	1	1.0	0.0	0.0	2	22	1	0.0	0.0	0.0
28	0.2	0.4	1	3	1	0.0	0.0	0.0	1	20	5	0.0	0.0	0.0
29	0.2	0.4	2	6	1	0.0	0.0	0.0	2	28	4	0.0	0.0	0.0
30	0.2	0.4	1	7	2	0.0	0.0	0.0	1	21	6	0.0	0.0	0.0
31	0.4	0.4	12	10	1	1.0	0.0	0.0	17	25	5	1.0	0.0	0.0
32	0.4	0.4	12	11	1	1.0	0.0	0.0	24	50	8	1.5	0.0	0.0
33	0.4	0.4	12	11	1	0.0	0.0	0.0	27	48	5	2.0	2.0	1.5
34	0.4	0.4	11	7	1	0.0	0.0	0.0	23	46	3	2.0	0.5	0.0
35	0.4	0.4	10	9	1	0.0	0.0	0.0	26	32	3	0.0	0.0	0.0
36	0.6	0.4	24	13	0	0.0	0.0	0.0	73	39	14	2.0	1.0	0.0
37	0.6	0.4	26	14	1	1.0	0.0	0.0	103	82	4	2.5	0.0	0.0
38	0.6	0.4	24	11	1	1.0	0.0	0.0	62	69	9	2.0	0.0	0.0
39	0.6	0.4	23	14	2	1.0	0.0	0.0	94	78	11	3.0	0.0	0.0
40	0.6	0.4	23	9	1	0.0	0.0	0.0	98	77	14	6.5	1.0	0.0
41	0.8	0.4	45	19	2	1.0	0.0	0.0	197	118	34	10.0	2.5	2.0
42	0.8	0.4	41	18	5	2.0	1.0	1.0	193	127	29	8.0	3.5	2.0
43	0.8	0.4	42	11	1	1.0	0.0	0.0	159	82	37	8.0	3.0	2.0
44	0.8	0.4	38	14	3	2.0	1.0	1.0	177	75	12	5.0	0.0	0.0
45	0.8	0.4	41	14	3	0.0	0.0	0.0	167	93	31	2.0	0.0	0.0
46	1.0	0.4	72	15	5	1.0	0.0	0.0	391	111	33	4.0	0.0	0.0
47	1.0	0.4	74	17	7	2.0	0.5	0.0	370	112	52	8.0	3.0	2.5
48	1.0	0.4	69	15	6	3.0	1.0	1.0	406	93	39	9.0	0.0	0.0
49	1.0	0.4	70	15	7	2.0	0.0	0.0	415	95	36	3.0	0.0	0.0
50	1.0	0.4	70	14	5	1.0	0.0	0.0	344	79	34	4.0	0.0	0.0
51	0.2	0.6	0	12	1	0.0	0.0	0.0	0	30	3	0.0	0.0	0.0
52	0.2	0.6	0	10	0	0.0	0.0	0.0	0	28	2	0.0	0.0	0.0
53	0.2	0.6	0	11	1	0.0	0.0	0.0	0	55	5	0.0	0.0	0.0
54	0.2	0.6	0	10	2	0.0	0.0	0.0	0	21	0	0.0	0.0	0.0
55	0.2	0.6	0	6	4	0.0	0.0	0.0	0	32	2	0.0	0.0	0.0
56	0.4	0.6	7	14	2	0.0	0.0	0.0	9	59	10	1.0	0.0	0.0
57	0.4	0.6	7	10	2	1.0	0.0	0.0	12	53	25	1.5	0.0	0.0
58	0.4	0.6	9	17	1	1.0	1.0	0.0	13	61	11	3.0	1.0	0.0
59	0.4	0.6	6	16	2	1.5	0.0	0.0	8	55	22	1.0	0.0	0.0
60	0.4	0.6	7	14	1	1.0	0.0	0.0	15	57	13	4.0	1.0	0.0
61	0.6	0.6	18	15	3	1.0	0.0	0.0	41	81	20	6.0	2.0	1.0
62	0.6	0.6	18	17	3	1.0	1.0	0.0	46	100	16	6.5	2.0	1.5
63	0.6	0.6	17	14	1	1.0	0.0	0.0	53	90	21	11.5	3.0	2.0
64	0.6	0.6	19	13	1	1.0	0.0	0.0	54	64	5	4.0	0.0	0.0
65	0.6	0.6	19	14	1	1.0	0.0	0.0	38	59	13	1.5	0.0	0.0
66	0.8	0.6	36	23	8	3.0	1.0	1.0	140	98	23	8.0	3.0	0.0
67	0.8	0.6	39	19	7	2.0	1.0	0.0	181	116	45	11.5	3.0	1.0
68	0.8	0.6	38	23	6	2.0	1.0	0.0	186	127	57	10.0	4.0	3.0
69	0.8	0.6	35	17	4	1.5	0.0	0.0	145	122	26	10.0	2.5	1.0
70	0.8	0.6	34	19	6	3.0	1.0	0.5	134	110	27	8.5	3.0	2.0
71	1.0	0.6	66	18	7	3.5	0.0	0.0	310	109	60	12.0	3.0	0.0
72	1.0	0.6	63	20	9	3.0	1.0	1.0	310	109	52	9.5	2.0	1.0
73	1.0	0.6	68	20	8	3.0	1.0	1.0	354	130	41	7.0	0.0	0.0
74	1.0	0.6	65	16	5	2.5	0.0	0.0	365	104	37	11.5	0.0	0.0
75	1.0	0.6	63	18	7	4.0	1.0	1.0	345	132	31	9.0	3.0	3.0
76	0.2	0.8	0	10	1	0.0	0.0	0.0	0	28	2	0.0	0.0	0.0
77	0.2	0.8	0	4	0	0.0	0.0	0.0	0	11	0	0.0	0.0	0.0
78	0.2	0.8	0	11	2	0.0	0.0	0.0	0	38	1	0.0	0.0	0.0
79	0.2	0.8	0	9	1	0.0	0.0	0.0	0	32	1	0.0	0.0	0.0
80	0.2	0.8	0	9	3	0.0	0.0	0.0	0	35	11	0.0	0.0	0.0
81	0.4	0.8	2	15	5	4.0	1.0	1.0	3	79	15	6.5	3.0	2.0
82	0.4	0.8	3	14	5	3.5	0.0	0.0	2	67	4	2.0	1.0	1.0
83	0.4	0.8	3	18	5	2.0	1.0	0.0	2	65	13	3.0	1.0	1.0
84	0.4	0.8	1	16	4	1.0	0.0	0.0	1	66	9	1.0	0.0	0.0
85	0.4	0.8	1	15	7	4.0	2.0	1.0	2	35	12	3.0	0.5	0.0
86	0.6	0.8	14	19	4	3.0	0.5	0.0	54	95	25	4.5	0.5	0.0
87	0.6	0.8	16	17	6	3.0	0.0	0.0	46	121	28	14.5	3.0	2.0
88	0.6	0.8	14	24	5	1.0	0.0	0.0	37	83	16	4.0	0.0	0.0
89	0.6	0.8	15	20	6	2.0	1.0	0.0	43	102	24	8.5	1.5	1.0
90	0.6	0.8	14	18	4	2.0	0.0	0.0	47	112	28	9.0	3.5	2.0

Table B.5(cntd) (a): $P3||\sum U_j$ (b): $P3||\sum w_jU_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
91	0.8	0.8	38	24	11	4.5	1.0	1.0	174	139	33	7.0	0.0	0.0
92	0.8	0.8	37	23	10	4.5	1.0	1.0	179	123	34	8.5	2.0	0.5
93	0.8	0.8	36	25	8	3.0	1.0	1.0	206	118	40	13.0	2.0	2.0
94	0.8	0.8	36	23	8	2.0	0.0	0.0	191	91	36	7.0	0.0	0.0
95	0.8	0.8	36	26	9	3.0	1.0	0.0	146	107	49	13.0	3.0	2.0
96	1.0	0.8	61	23	12	4.0	1.0	1.0	297	113	50	14.0	4.0	1.0
97	1.0	0.8	60	15	8	3.0	0.0	0.0	339	131	56	8.0	0.0	0.0
98	1.0	0.8	64	21	7	4.5	1.0	0.0	356	104	29	6.0	0.0	0.0
99	1.0	0.8	64	19	10	5.0	2.0	1.0	370	148	68	17.0	1.0	1.0
100	1.0	0.8	54	20	8	4.0	1.0	0.0	271	104	39	13.0	1.0	0.5
101	0.2	1.0	0	7	1	0.0	0.0	0.0	0	22	0	0.0	0.0	0.0
102	0.2	1.0	0	2	0	0.0	0.0	0.0	0	10	0	0.0	0.0	0.0
103	0.2	1.0	0	8	0	0.0	0.0	0.0	0	25	0	0.0	0.0	0.0
104	0.2	1.0	0	12	0	0.0	0.0	0.0	0	27	0	0.0	0.0	0.0
105	0.2	1.0	0	5	0	0.0	0.0	0.0	0	15	2	0.0	0.0	0.0
106	0.4	1.0	0	13	2	2.0	0.0	0.0	0	25	4	0.0	0.0	0.0
107	0.4	1.0	3	20	4	3.0	1.0	0.0	3	48	15	5.0	1.0	0.0
108	0.4	1.0	1	15	4	3.0	1.0	0.0	0	51	17	3.0	1.0	1.0
109	0.4	1.0	2	16	5	4.0	1.0	0.5	2	53	17	8.0	1.5	1.0
110	0.4	1.0	1	14	5	4.0	1.0	1.0	1	61	16	5.0	1.0	1.0
111	0.6	1.0	20	22	7	2.0	0.0	0.0	80	80	22	6.5	1.0	0.0
112	0.6	1.0	21	22	7	2.0	1.0	0.0	100	120	39	11.5	3.0	3.0
113	0.6	1.0	14	21	7	4.0	1.0	1.0	55	103	31	10.5	2.0	2.0
114	0.6	1.0	18	25	9	4.0	1.0	1.0	89	118	46	11.0	3.5	1.0
115	0.6	1.0	14	24	7	3.0	1.0	0.0	67	119	26	6.5	0.0	0.0
116	0.8	1.0	41	27	10	5.0	0.5	0.0	200	132	42	17.0	4.0	3.0
117	0.8	1.0	37	27	11	5.0	1.0	0.0	219	129	35	10.5	1.0	1.0
118	0.8	1.0	38	24	9	5.0	1.5	1.0	159	98	33	14.5	3.0	1.0
119	0.8	1.0	37	21	6	5.0	0.0	0.0	181	119	39	17.0	0.0	0.0
120	0.8	1.0	34	21	9	5.0	1.0	1.0	178	72	16	6.0	1.0	1.0
121	1.0	1.0	52	16	9	4.0	1.0	0.0	261	124	43	13.0	1.5	0.5
122	1.0	1.0	56	27	12	6.0	2.0	1.0	317	161	52	4.5	1.0	1.0
123	1.0	1.0	49	21	10	5.0	1.0	1.0	249	108	38	6.0	3.0	1.5
124	1.0	1.0	52	20	9	4.0	0.0	0.0	256	99	31	4.5	0.0	0.0
125	1.0	1.0	52	19	8	3.0	0.0	0.0	267	115	45	12.5	2.0	1.0

Table B.6 (a): $P3||\sum T_j$ (b): $P3||\sum w_jT_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	837	1258	873	155	155	155	2458	2228	559	2	0	0
2	0.2	0.2	732	1196	727	217	217	217	2522	2229	903	16	0	0
3	0.2	0.2	830	1214	879	183	181	181	1733	1915	348	12	0	0
4	0.2	0.2	673	921	516	122	122	122	2043	3103	839	0	0	0
5	0.2	0.2	1102	1139	843	48	48	48	2134	1942	698	28	0	0
6	0.4	0.2	6061	2202	1324	489	486	486	21482	6004	2848	38	0	0
7	0.4	0.2	5579	2416	1708	530	505	505	18695	7083	3492	8	0	0
8	0.4	0.2	6256	2262	1603	539	539	539	21623	4708	2402	20	7	7
9	0.4	0.2	6123	2916	1949	572	545	545	14942	6882	2202	4	0	0
10	0.4	0.2	5513	1517	1095	514	514	514	19412	4279	1752	88	0	0
11	0.6	0.2	15768	2441	1556	673	629	629	64743	9357	4927	481	12	2
12	0.6	0.2	17704	3116	2347	930	911	911	82757	12809	6677	473	43	24
13	0.6	0.2	19385	2613	1378	488	471	468	63744	10467	5401	314	4	1
14	0.6	0.2	17007	1563	871	16	0	0	56079	6451	4161	134	6	2
15	0.6	0.2	14065	2857	1804	799	792	792	61699	6885	3435	290	2	0
16	0.8	0.2	34246	2630	1481	722	672	668	143149	13247	5572	662	28	14
17	0.8	0.2	31822	2576	1686	1200	1182	1182	117027	9949	4329	411	39	20
18	0.8	0.2	34539	2640	1708	959	912	910	189935	10442	2915	509	20	12
19	0.8	0.2	34760	1903	1041	22	4	4	167157	10444	3325	400	78	36
20	0.8	0.2	30457	3208	1899	790	762	762	142258	12561	5119	448	29	12
21	1.0	0.2	65533	1952	1683	1652	1646	1646	310829	2068	466	92	10	6
22	1.0	0.2	45253	1455	1309	1294	1292	1292	193268	1446	402	68	24	11
23	1.0	0.2	45829	1145	1001	931	928	927	187583	1425	431	62	11	2
24	1.0	0.2	51185	273	99	80	78	78	257734	1185	189	54	13	5
25	1.0	0.2	47119	771	613	577	572	571	203705	1085	149	62	16	8
26	0.2	0.4	33	1313	392	0	0	0	99	4341	1373	24	0	0
27	0.2	0.4	229	1619	428	4	0	0	440	2580	566	5	0	0
28	0.2	0.4	0	1370	242	22	22	22	100	4091	1253	16	0	0
29	0.2	0.4	140	2047	546	24	24	24	356	5299	672	56	0	0
30	0.2	0.4	15	2707	405	12	12	12	48	4437	604	17	0	0
31	0.4	0.4	4061	4249	1852	214	203	203	8981	9268	2983	132	0	0
32	0.4	0.4	4218	5036	1775	228	182	182	9767	18317	7890	164	0	0
33	0.4	0.4	4725	4269	2006	189	145	144	12431	16779	5615	152	1	0
34	0.4	0.4	4160	3448	1571	316	308	308	8597	10517	3021	350	11	8
35	0.4	0.4	2536	3499	1454	236	232	232	7491	7527	4542	65	6	0
36	0.6	0.4	13355	4996	2022	512	480	478	39386	18789	5620	371	13	4
37	0.6	0.4	17413	5511	2577	760	692	684	64916	36241	15721	447	39	11
38	0.6	0.4	14333	5058	3088	786	681	675	33148	22813	6475	224	23	6
39	0.6	0.4	13225	5398	2565	581	561	560	54590	25541	9245	770	20	0

Table B.6(cntd) (a): $P3||\sum T_j$ (b): $P3||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
40	0.6	0.4	13206	4623	2068	72	18	12	46702	30068	12233	380	52	32
41	0.8	0.4	43208	4433	2040	1246	1209	1204	163400	19370	3267	492	87	44
42	0.8	0.4	33257	4137	1515	870	837	832	149615	15104	2773	476	59	18
43	0.8	0.4	34933	3579	1431	853	762	754	113577	15881	3866	584	128	78
44	0.8	0.4	28874	3851	1475	600	564	558	127405	19422	5101	802	109	46
45	0.8	0.4	30973	4470	1699	1044	988	982	109547	21589	3847	456	81	45
46	1.0	0.4	57141	2137	1560	1440	1436	1435	287286	4038	916	105	20	10
47	1.0	0.4	49964	1074	667	633	631	631	216486	2927	454	49	18	8
48	1.0	0.4	47467	1690	1185	1048	1038	1038	259899	3890	1210	163	22	10
49	1.0	0.4	46994	645	145	75	60	60	228194	2942	849	98	22	12
50	1.0	0.4	48630	1757	1355	1277	1272	1271	208179	4117	1565	129	47	38
51	0.2	0.6	0	2568	0	0	0	0	0	8993	597	0	0	0
52	0.2	0.6	0	1702	0	0	0	0	0	4240	0	0	0	0
53	0.2	0.6	0	2815	0	0	0	0	0	13894	0	0	0	0
54	0.2	0.6	0	3425	469	0	0	0	0	5132	98	0	0	0
55	0.2	0.6	0	4057	612	0	0	0	0	9772	0	0	0	0
56	0.4	0.6	1820	6425	2427	150	122	120	3680	17295	3506	278	0	0
57	0.4	0.6	1127	9032	3340	97	94	94	4667	32174	8007	21	0	0
58	0.4	0.6	3031	7504	2442	180	138	134	6450	22180	5980	709	24	19
59	0.4	0.6	1210	6422	1757	128	103	103	3114	22562	5443	2	0	0
60	0.4	0.6	2176	7681	2678	158	145	145	7689	33719	9019	80	0	0
61	0.6	0.6	14352	7263	2357	601	482	469	31838	33064	7086	674	76	41
62	0.6	0.6	14004	8922	2847	216	102	84	32196	43924	12166	808	152	50
63	0.6	0.6	13426	6909	2324	187	48	22	36942	40732	9841	864	122	83
64	0.6	0.6	12739	8015	2804	86	8	2	36618	25659	6873	236	3	2
65	0.6	0.6	9433	8811	3200	554	441	426	20349	28183	8558	634	26	8
66	0.8	0.6	26073	4025	741	313	276	269	87504	20571	3823	450	156	86
67	0.8	0.6	31229	4487	813	64	16	11	141964	27564	3679	650	143	80
68	0.8	0.6	32843	5165	658	256	216	206	141817	27178	4315	611	122	54
69	0.8	0.6	29503	4224	927	495	436	430	110809	25045	3966	707	157	54
70	0.8	0.6	23140	4704	985	296	233	225	80192	22765	3632	768	130	62
71	1.0	0.6	52157	2240	1023	874	854	851	222867	6734	944	153	34	11
72	1.0	0.6	45000	2314	1370	1191	1184	1182	213015	7670	821	152	32	13
73	1.0	0.6	47800	2100	1140	1096	1088	1086	217323	3540	243	74	26	15
74	1.0	0.6	42487	2022	522	258	242	240	203391	7637	967	170	33	22
75	1.0	0.6	40377	2164	1237	1120	1108	1104	200751	4629	351	74	16	4
76	0.2	0.8	0	4252	10	0	0	0	0	10910	0	0	0	0
77	0.2	0.8	0	1302	0	0	0	0	0	4653	0	0	0	0
78	0.2	0.8	0	5875	67	0	0	0	0	12928	99	0	0	0
79	0.2	0.8	0	4934	0	0	0	0	0	13045	74	0	0	0
80	0.2	0.8	0	4270	179	0	0	0	0	13997	0	0	0	0
81	0.4	0.8	363	9026	2236	79	16	7	883	35662	7552	346	148	120
82	0.4	0.8	271	11150	2353	44	19	19	330	22919	2336	142	7	4
83	0.4	0.8	277	10582	2617	14	2	1	663	38188	4742	276	208	6
84	0.4	0.8	34	7014	895	33	33	33	178	26191	3679	7	0	0
85	0.4	0.8	105	11888	2652	62	36	36	314	32164	8795	127	4	4
86	0.6	0.8	10325	11363	3148	390	292	258	26434	45679	10571	1350	207	109
87	0.6	0.8	13003	10597	2190	176	48	20	33118	52915	11364	1530	286	113
88	0.6	0.8	9846	9702	1207	283	174	143	22238	42214	7913	1104	186	64
89	0.6	0.8	8674	8639	2486	312	190	176	20956	32921	7639	468	268	17
90	0.6	0.8	11751	9920	2032	378	206	180	30437	47683	7988	1030	128	42
91	0.8	0.8	27429	4966	413	50	12	4	90313	24163	4718	510	71	52
92	0.8	0.8	27975	5967	1120	335	270	256	111378	35278	5007	672	132	72
93	0.8	0.8	25896	4709	1096	582	540	532	116143	27281	3574	292	82	38
94	0.8	0.8	24873	5350	1294	472	426	416	98415	27968	4446	394	170	54
95	0.8	0.8	32392	6217	1291	676	615	608	113547	41325	6431	666	134	80
96	1.0	0.8	43728	1974	1126	1020	1013	1013	173763	9591	1008	176	32	18
97	1.0	0.8	44211	2752	1293	1077	1060	1056	222245	6877	704	136	34	28
98	1.0	0.8	53768	1656	514	428	418	417	236116	5762	442	118	20	14
99	1.0	0.8	42786	2337	630	498	489	488	217223	7761	782	154	30	18
100	1.0	0.8	39841	2853	702	352	328	322	157442	11613	1897	241	71	33
101	0.2	1.0	0	3319	0	0	0	0	0	10252	0	0	0	0
102	0.2	1.0	0	4547	0	0	0	0	0	10692	0	0	0	0
103	0.2	1.0	0	3240	0	0	0	0	0	14687	0	0	0	0
104	0.2	1.0	0	4593	0	0	0	0	0	16819	44	0	0	0
105	0.2	1.0	0	3367	0	0	0	0	0	11281	0	0	0	0
106	0.4	1.0	0	8206	138	0	0	0	0	21118	455	0	0	0
107	0.4	1.0	315	15347	2848	15	0	0	666	40917	8233	112	0	0
108	0.4	1.0	0	11392	1592	0	0	0	0	33482	3539	0	0	0
109	0.4	1.0	47	12016	2649	25	0	0	223	39655	6707	314	0	0
110	0.4	1.0	3	11286	2339	8	0	0	6	43913	7859	70	0	0
111	0.6	1.0	17951	10868	902	146	39	15	59210	67169	15220	1001	202	64
112	0.6	1.0	17413	10306	1674	198	52	14	64874	51458	7440	794	178	94
113	0.6	1.0	12191	13678	2182	270	136	118	35673	64329	12255	944	190	90
114	0.6	1.0	17448	10752	929	154	40	14	62947	50506	9141	853	193	108
115	0.6	1.0	8713	11057	1454	310	185	160	28230	55848	8528	884	242	124
116	0.8	1.0	31149	5168	1027	498	452	436	131530	29612	4860	369	94	52
117	0.8	1.0	26244	6028	1124	568	506	500	116615	35975	4857	423	133	54
118	0.8	1.0	28183	6195	1052	222	150	138	88534	34458	3956	751	119	62
119	0.8	1.0	26997	5804	890	136	64	53	105718	34250	8191	546	137	64
120	0.8	1.0	27711	6346	971	90	20	6	95521	32896	3137	592	108	53

Table B.6(cntd) (a): $P3||\sum T_j$ (b): $P3||\sum w_jT_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
121	1.0	1.0	40696	3747	454	41	13	4	165788	21426	2718	470	86	48
122	1.0	1.0	46571	2940	942	880	866	865	199631	11511	455	128	26	7
123	1.0	1.0	34533	4408	1298	830	809	802	140244	14380	2528	287	63	28
124	1.0	1.0	41680	3394	653	364	328	323	151956	21135	4036	380	146	71
125	1.0	1.0	44121	4682	1377	914	860	851	196206	24889	3382	486	97	50

Table B.7 (a): $P4||\sum U_j$ (b): $P4||\sum w_jU_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	6	4	1	1.0	0.0	0.0	9	27	2	1.0	0.0	0.0
2	0.2	0.2	6	5	1	1.0	0.0	0.0	12	20	5	1.5	0.0	0.0
3	0.2	0.2	7	6	1	0.0	0.0	0.0	8	22	3	1.0	0.0	0.0
4	0.2	0.2	6	7	1	0.0	0.0	0.0	12	26	5	0.5	0.0	0.0
5	0.2	0.2	6	7	2	1.0	0.0	0.0	10	29	3	1.0	0.0	0.0
6	0.4	0.2	18	6	1	0.5	0.0	0.0	67	40	7	2.0	1.0	0.5
7	0.4	0.2	17	10	1	0.0	0.0	0.0	55	33	5	2.0	0.5	0.0
8	0.4	0.2	18	9	2	1.0	1.0	1.0	54	42	4	1.0	0.0	0.0
9	0.4	0.2	18	9	1	0.0	0.0	0.0	44	46	4	1.0	0.0	0.0
10	0.4	0.2	17	7	1	1.0	0.0	0.0	54	31	4	2.0	1.0	1.0
11	0.6	0.2	30	8	1	0.0	0.0	0.0	130	56	5	2.0	1.0	1.0
12	0.6	0.2	32	9	1	0.0	0.0	0.0	156	71	5	3.0	0.0	0.0
13	0.6	0.2	32	14	2	1.0	0.0	0.0	129	60	5	3.0	1.0	1.0
14	0.6	0.2	31	8	1	0.0	0.0	0.0	118	30	7	4.0	1.5	1.0
15	0.6	0.2	29	10	1	1.0	0.0	0.0	136	52	7	4.0	1.0	1.0
16	0.8	0.2	48	11	1	1.0	0.5	0.0	245	79	17	3.5	1.0	1.0
17	0.8	0.2	47	14	1	1.0	0.0	0.0	198	69	6	2.0	1.0	1.0
18	0.8	0.2	49	12	1	0.0	0.0	0.0	282	66	10	4.0	0.0	0.0
19	0.8	0.2	47	12	2	2.0	1.0	1.0	253	112	12	8.0	4.0	2.0
20	0.8	0.2	45	9	1	0.0	0.0	0.0	228	75	8	4.5	1.5	1.0
21	1.0	0.2	82	9	3	0.0	0.0	0.0	467	57	16	3.0	0.0	0.0
22	1.0	0.2	80	10	2	1.0	1.0	1.0	398	44	14	4.0	3.0	2.0
23	1.0	0.2	80	9	4	2.0	1.0	0.5	403	62	13	5.0	1.0	1.0
24	1.0	0.2	82	9	2	0.0	0.0	0.0	472	67	10	6.0	0.0	0.0
25	1.0	0.2	80	10	1	0.5	0.0	0.0	441	73	13	3.0	1.0	1.0
26	0.2	0.4	1	8	2	1.0	0.0	0.0	1	38	5	1.0	0.0	0.0
27	0.2	0.4	3	9	0	0.0	0.0	0.0	3	33	2	0.0	0.0	0.0
28	0.2	0.4	1	9	1	1.0	0.0	0.0	1	28	4	1.0	0.0	0.0
29	0.2	0.4	2	11	2	1.0	0.0	0.0	3	27	4	1.0	0.5	0.0
30	0.2	0.4	1	13	2	1.0	0.0	0.0	1	30	2	1.0	0.0	0.0
31	0.4	0.4	13	13	0	0.0	0.0	0.0	18	44	7	1.5	0.0	0.0
32	0.4	0.4	12	16	1	1.0	1.0	1.0	26	70	7	3.0	0.5	0.0
33	0.4	0.4	12	16	1	1.0	0.0	0.0	29	69	6	5.0	1.0	0.0
34	0.4	0.4	12	9	1	1.0	0.0	0.0	25	74	5	3.0	1.0	1.0
35	0.4	0.4	10	13	2	1.0	0.0	0.0	26	44	6	3.0	1.0	1.0
36	0.6	0.4	24	19	2	1.0	0.0	0.0	74	62	9	3.5	1.0	1.0
37	0.6	0.4	27	18	0	0.0	0.0	0.0	107	111	4	1.5	1.0	1.0
38	0.6	0.4	25	10	1	0.0	0.0	0.0	63	92	9	4.5	2.0	1.0
39	0.6	0.4	24	17	1	1.0	0.0	0.0	97	126	19	5.0	0.0	0.0
40	0.6	0.4	23	14	2	1.0	1.0	1.0	102	140	14	7.0	2.0	0.0
41	0.8	0.4	46	23	3	1.5	1.0	1.0	207	146	23	8.5	3.0	1.5
42	0.8	0.4	42	23	2	1.0	0.0	0.0	197	151	26	10.0	3.5	2.0
43	0.8	0.4	42	13	1	1.0	1.0	0.0	166	112	15	5.0	1.0	0.0
44	0.8	0.4	39	17	2	1.0	0.0	0.0	185	76	14	4.5	1.0	0.0
45	0.8	0.4	42	16	1	1.0	0.0	0.0	177	109	9	4.0	2.0	0.0
46	1.0	0.4	73	14	4	1.0	1.0	1.0	400	102	17	5.0	2.0	0.0
47	1.0	0.4	75	17	3	1.0	0.0	0.0	376	115	37	5.0	2.0	2.0
48	1.0	0.4	70	16	4	2.0	1.0	1.0	415	103	29	3.0	0.0	0.0
49	1.0	0.4	72	13	2	1.0	0.0	0.0	427	93	25	4.0	1.0	0.0
50	1.0	0.4	71	15	2	1.0	0.0	0.0	349	82	15	3.0	0.0	0.0
51	0.2	0.6	0	15	0	0.0	0.0	0.0	0	40	2	0.0	0.0	0.0
52	0.2	0.6	0	14	0	0.0	0.0	0.0	0	30	0	0.0	0.0	0.0
53	0.2	0.6	0	13	0	0.0	0.0	0.0	0	67	0	0.0	0.0	0.0
54	0.2	0.6	0	10	2	0.0	0.0	0.0	0	25	0	0.0	0.0	0.0
55	0.2	0.6	0	11	2	0.0	0.0	0.0	0	44	1	0.0	0.0	0.0
56	0.4	0.6	7	20	2	1.0	0.0	0.0	10	89	5	2.5	0.0	0.0
57	0.4	0.6	7	17	2	1.0	0.0	0.0	13	82	10	4.0	1.0	1.0
58	0.4	0.6	9	21	2	1.5	0.5	0.0	14	72	9	4.0	1.0	0.0
59	0.4	0.6	7	17	2	0.0	0.0	0.0	9	82	11	2.0	0.0	0.0
60	0.4	0.6	8	18	1	1.0	0.0	0.0	17	96	11	5.0	1.5	1.0
61	0.6	0.6	18	18	2	2.0	1.0	1.0	44	100	7	7.0	0.5	0.0
62	0.6	0.6	19	23	1	1.0	0.0	0.0	48	142	13	7.0	2.0	2.0
63	0.6	0.6	18	22	1	1.0	0.0	0.0	59	112	26	12.0	3.0	1.5
64	0.6	0.6	20	20	2	1.0	0.0	0.0	57	109	9	5.0	1.0	1.0
65	0.6	0.6	20	18	1	1.0	0.0	0.0	40	99	6	3.0	0.0	0.0
66	0.8	0.6	38	23	5	2.0	0.0	0.0	144	116	18	9.0	3.5	2.5
67	0.8	0.6	40	20	4	1.5	0.0	0.0	188	127	29	11.5	3.0	2.0
68	0.8	0.6	39	22	4	2.0	0.5	0.0	192	137	26	9.0	3.0	1.0
69	0.8	0.6	36	22	3	1.5	1.0	0.5	154	145	26	11.0	2.0	2.0

Table B.7(cntd) (a): $P4||\sum U_j$ (b): $P4||\sum w_jU_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
70	0.8	0.6	35	21	5	2.0	1.0	1.0	138	142	22	7.5	3.0	2.0
71	1.0	0.6	67	17	3	1.0	0.0	0.0	321	108	24	7.0	1.0	1.0
72	1.0	0.6	65	18	4	2.0	1.0	0.0	332	96	22	5.0	1.0	0.0
73	1.0	0.6	69	19	4	1.0	1.0	1.0	359	125	17	6.0	2.0	1.0
74	1.0	0.6	66	15	3	1.0	0.0	0.0	380	99	20	3.0	0.0	0.0
75	1.0	0.6	64	18	3	1.0	1.0	0.0	351	126	17	4.5	2.0	2.0
76	0.2	0.8	0	16	0	0.0	0.0	0.0	0	47	0	0.0	0.0	0.0
77	0.2	0.8	0	8	0	0.0	0.0	0.0	0	36	0	0.0	0.0	0.0
78	0.2	0.8	0	15	0	0.0	0.0	0.0	0	49	0	0.0	0.0	0.0
79	0.2	0.8	0	14	0	0.0	0.0	0.0	0	38	0	0.0	0.0	0.0
80	0.2	0.8	0	11	1	0.0	0.0	0.0	0	45	0	0.0	0.0	0.0
81	0.4	0.8	4	18	2	2.0	0.0	0.0	8	117	14	9.0	1.0	0.5
82	0.4	0.8	3	23	4	2.0	0.5	0.0	3	110	3	2.0	1.0	1.0
83	0.4	0.8	3	26	4	3.0	1.0	1.0	4	105	11	3.0	0.0	0.0
84	0.4	0.8	1	20	3	2.0	1.0	1.0	2	98	6	3.0	0.5	0.0
85	0.4	0.8	3	23	3	2.0	0.0	0.0	3	74	15	6.0	2.0	1.0
86	0.6	0.8	15	27	3	1.0	0.0	0.0	60	112	19	6.5	2.0	1.0
87	0.6	0.8	16	23	4	2.0	1.0	0.0	54	139	32	16.0	6.0	2.0
88	0.6	0.8	15	29	2	1.5	0.5	0.0	40	91	11	8.0	1.0	1.0
89	0.6	0.8	16	25	3	1.0	0.0	0.0	52	135	13	7.0	2.0	1.0
90	0.6	0.8	15	19	3	1.0	1.0	0.5	58	143	16	10.0	4.0	2.5
91	0.8	0.8	39	26	6	3.0	1.5	1.0	184	134	22	5.0	2.0	1.0
92	0.8	0.8	38	25	6	3.0	1.0	1.0	181	123	27	8.0	2.0	0.0
93	0.8	0.8	37	28	2	2.0	1.0	1.0	220	104	17	9.5	2.5	2.0
94	0.8	0.8	37	23	6	2.0	0.0	0.0	192	114	24	7.5	1.0	0.0
95	0.8	0.8	37	27	5	2.0	1.0	0.0	150	120	31	12.0	1.0	1.0
96	1.0	0.8	62	24	4	2.0	1.0	1.0	309	123	28	8.0	0.0	0.0
97	1.0	0.8	61	16	3	1.5	0.0	0.0	356	124	10	1.0	0.0	0.0
98	1.0	0.8	65	20	4	2.0	1.0	1.0	369	91	7	2.0	0.5	0.0
99	1.0	0.8	65	19	5	2.0	0.0	0.0	382	136	31	6.5	0.0	0.0
100	1.0	0.8	55	19	3	2.0	1.0	0.0	279	104	20	5.0	1.0	1.0
101	0.2	1.0	0	11	0	0.0	0.0	0.0	0	49	0	0.0	0.0	0.0
102	0.2	1.0	0	8	0	0.0	0.0	0.0	0	34	0	0.0	0.0	0.0
103	0.2	1.0	0	13	0	0.0	0.0	0.0	0	59	0	0.0	0.0	0.0
104	0.2	1.0	0	15	0	0.0	0.0	0.0	0	50	2	0.0	0.0	0.0
105	0.2	1.0	0	13	0	0.0	0.0	0.0	0	40	0	0.0	0.0	0.0
106	0.4	1.0	0	22	3	2.0	0.0	0.0	0	52	3	1.0	0.0	0.0
107	0.4	1.0	4	29	4	3.0	1.0	0.0	5	78	10	7.0	1.5	1.0
108	0.4	1.0	1	21	4	4.0	1.0	1.0	2	90	14	9.0	2.0	1.0
109	0.4	1.0	2	23	3	3.0	2.0	1.0	5	110	11	9.0	3.0	1.0
110	0.4	1.0	2	21	4	4.0	1.0	1.0	3	95	17	9.0	1.0	0.0
111	0.6	1.0	20	27	4	2.0	1.0	0.0	89	114	16	8.5	3.0	2.0
112	0.6	1.0	21	27	7	3.0	1.0	1.0	103	136	31	13.0	6.0	3.0
113	0.6	1.0	15	25	4	3.0	1.0	1.0	70	126	17	9.5	1.0	1.0
114	0.6	1.0	19	28	4	3.0	1.0	0.5	96	122	24	9.5	3.0	1.0
115	0.6	1.0	15	25	4	2.0	1.0	0.0	78	121	8	2.0	1.0	1.0
116	0.8	1.0	42	27	7	3.5	1.0	1.0	217	115	30	8.5	2.0	0.0
117	0.8	1.0	38	27	6	3.0	1.0	0.0	225	136	16	3.5	0.0	0.0
118	0.8	1.0	40	24	6	3.0	1.0	1.0	168	105	17	9.5	2.0	0.0
119	0.8	1.0	37	23	3	2.0	1.0	0.0	181	145	28	15.5	1.0	1.0
120	0.8	1.0	36	22	5	2.0	1.0	0.0	186	64	8	3.0	0.0	0.0
121	1.0	1.0	52	19	5	3.0	1.0	1.0	266	131	28	8.0	1.0	1.0
122	1.0	1.0	58	27	5	3.0	1.0	1.0	331	147	18	3.0	1.0	0.5
123	1.0	1.0	50	20	5	2.0	1.0	0.5	252	122	13	2.0	0.0	0.0
124	1.0	1.0	52	23	6	3.0	1.0	0.0	257	103	23	5.0	1.0	1.0
125	1.0	1.0	52	20	4	2.0	0.5	0.0	276	116	23	7.5	1.0	0.5

Table B.8 (a): $P4||\sum T_j$ (b): $P4||\sum w_jT_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	824	1034	593	0.0	0.0	0.0	2001	2847	476	4	0	0.0
2	0.2	0.2	802	869	380	0.0	0.0	0.0	2080	2406	775	12	0	0.0
3	0.2	0.2	839	1009	587	0.0	0.0	0.0	1424	1740	204	14	0	0.0
4	0.2	0.2	675	724	247	0.0	0.0	0.0	1683	3000	908	0	0	0.0
5	0.2	0.2	958	934	475	5.0	0.0	0.0	1727	1726	575	0	0	0.0
6	0.4	0.2	5138	1427	747	3.5	0.0	0.0	16893	5151	2446	83	0	0.0
7	0.4	0.2	4779	1577	1033	34.0	0.0	0.0	14659	5319	2287	37	0	0.0
8	0.4	0.2	5321	1523	667	3.0	0.0	0.0	16841	3956	1996	16	0	0.0
9	0.4	0.2	5243	1963	1003	26.0	0.0	0.0	11753	5658	1832	22	0	0.0
10	0.4	0.2	4744	956	614	15.0	0.0	0.0	15226	3877	1694	74	0	0.0
11	0.6	0.2	12664	1477	629	37.0	1.0	0.0	50234	7506	4102	282	17	9.5
12	0.6	0.2	14339	1794	978	39.0	0.0	0.0	63837	9731	5446	272	37	25.5
13	0.6	0.2	15274	1788	683	21.5	3.0	0.0	49438	8234	3140	254	6	0.5
14	0.6	0.2	13098	1216	721	53.5	0.0	0.0	43406	4917	2820	184	9	3.0
15	0.6	0.2	11469	1686	886	17.5	1.0	0.0	47892	6852	2969	254	2	1.0
16	0.8	0.2	26720	1465	650	40.0	6.5	6.0	110236	11036	4509	328	40	22.5
17	0.8	0.2	25250	1213	488	48.5	4.0	1.0	90114	8292	2276	211	32	22.0
18	0.8	0.2	27123	1271	433	32.0	9.0	5.0	145609	7829	2941	646	98	14.5

Table B.8(cntd) (a): $P4||\sum T_j$ (b): $P4||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
19	0.8	0.2	26603	1546	549	24.0	2.5	0.0	128341	8147	3322	423	72	65.0
20	0.8	0.2	23900	1737	645	65.0	4.0	1.0	109345	9953	3515	708	44	28.0
21	1.0	0.2	51113	232	43	6.5	1.0	1.0	237394	1618	375	54	16	7.5
22	1.0	0.2	35465	128	23	2.0	0.0	0.0	147842	1139	185	46	18	8.5
23	1.0	0.2	35634	174	92	6.0	0.0	0.0	143598	1107	282	58	11	5.0
24	1.0	0.2	39058	133	22	3.0	0.0	0.0	196978	921	223	68	18	11.5
25	1.0	0.2	36369	150	30	1.0	0.5	0.0	156044	853	123	57	12	4.5
26	0.2	0.4	47	1427	423	0.0	0.0	0.0	141	3828	891	82	0	0.0
27	0.2	0.4	212	1365	372	0.5	0.0	0.0	388	2862	560	47	0	0.0
28	0.2	0.4	26	1337	310	1.0	0.0	0.0	112	3977	1122	2	0	0.0
29	0.2	0.4	163	1862	544	0.0	0.0	0.0	350	4684	1053	59	2	0.0
30	0.2	0.4	36	2265	405	2.0	0.0	0.0	51	3864	755	9	0	0.0
31	0.4	0.4	3407	3603	1318	15.5	1.0	0.0	7130	7481	2158	21	0	0.0
32	0.4	0.4	3517	4039	1647	13.0	0.0	0.0	7730	13122	3992	326	0	0.0
33	0.4	0.4	3875	3487	1550	42.0	15.0	7.5	9965	14050	4414	387	5	0.0
34	0.4	0.4	3556	2810	1195	13.0	0.0	0.0	6927	9187	1689	333	13	6.0
35	0.4	0.4	2227	2772	874	12.0	0.0	0.0	6054	6724	1991	236	0	0.0
36	0.6	0.4	10727	3497	1314	60.0	4.0	0.5	30797	15441	4550	267	90	26.5
37	0.6	0.4	13993	3941	1217	87.5	14.0	9.0	50313	23150	7559	622	77	16.5
38	0.6	0.4	11626	3511	1685	74.0	6.0	2.5	25902	18543	6363	374	10	9.0
39	0.6	0.4	10676	3729	1322	54.0	8.5	3.0	42346	20838	6411	518	38	15.5
40	0.6	0.4	10236	3800	1722	89.5	10.0	3.5	36345	24457	7478	612	39	32.0
41	0.8	0.4	33984	2509	522	48.0	8.0	4.0	126234	15228	4029	476	72	21.5
42	0.8	0.4	26138	2745	589	50.5	16.0	10.5	115091	12411	2556	454	110	66.5
43	0.8	0.4	27354	2299	579	60.5	18.0	11.5	87902	12158	2704	302	44	18.0
44	0.8	0.4	22648	2527	627	73.5	16.0	8.5	98472	14328	3645	432	74	49.0
45	0.8	0.4	24546	2826	654	46.0	14.0	7.0	84960	17284	3606	500	126	61.5
46	1.0	0.4	44625	543	87	5.5	1.0	1.0	219564	3156	481	108	34	14.5
47	1.0	0.4	38562	340	27	4.0	0.0	0.0	165648	2161	556	92	24	13.0
48	1.0	0.4	37023	464	136	6.0	1.5	0.0	198747	3049	758	143	54	30.0
49	1.0	0.4	35897	475	71	4.0	0.0	0.0	174710	2324	534	92	22	12.0
50	1.0	0.4	38035	388	42	10.0	1.5	0.5	159335	3055	707	118	28	14.5
51	0.2	0.6	0	2324	0	0.0	0.0	0.0	0	7021	426	0	0	0.0
52	0.2	0.6	0	1436	0	0.0	0.0	0.0	0	3988	0	0	0	0.0
53	0.2	0.6	0	2263	0	0.0	0.0	0.0	0	11069	0	0	0	0.0
54	0.2	0.6	0	2671	353	0.0	0.0	0.0	0	4147	0	0	0	0.0
55	0.2	0.6	0	3224	215	0.0	0.0	0.0	0	8567	21	0	0	0.0
56	0.4	0.6	1625	4839	1509	28.0	5.0	5.0	3076	15682	1894	73	0	0.0
57	0.4	0.6	1021	7202	2618	8.0	0.0	0.0	3811	24660	5797	134	0	0.0
58	0.4	0.6	2606	6046	1708	69.0	8.0	3.5	5288	19458	4953	494	124	21.5
59	0.4	0.6	1113	5247	1337	43.0	3.0	0.0	2545	19265	3608	22	0	0.0
60	0.4	0.6	1903	6430	1795	33.5	4.0	2.0	6276	26120	6538	163	0	0.0
61	0.6	0.6	11543	5392	1718	117.0	23.5	9.5	24912	22803	5604	922	117	50.0
62	0.6	0.6	10982	6988	2157	165.5	27.5	7.5	25405	35232	8538	1171	108	50.0
63	0.6	0.6	10505	5401	1692	146.5	27.0	17.0	29390	32886	7809	1020	76	21.0
64	0.6	0.6	9928	6385	2252	51.0	11.5	7.5	28604	21767	5693	489	10	0.0
65	0.6	0.6	7703	6740	2034	193.5	25.5	11.0	15865	22560	6444	580	12	3.0
66	0.8	0.6	20289	2957	513	35.0	14.5	6.5	68044	15865	3633	342	87	50.0
67	0.8	0.6	23974	3554	699	68.5	22.5	11.5	109608	21946	3057	710	142	92.5
68	0.8	0.6	25402	3824	917	66.0	18.5	7.0	109692	21754	2597	497	117	68.5
69	0.8	0.6	22998	3037	603	61.0	26.5	9.5	86216	19364	4321	670	124	77.0
70	0.8	0.6	18029	3575	725	84.0	27.0	8.0	62498	18129	4495	456	85	31.5
71	1.0	0.6	40414	1123	187	16.0	2.5	1.0	170726	5015	705	158	38	16.5
72	1.0	0.6	35281	853	168	10.5	1.5	0.5	163264	5995	1155	185	32	15.0
73	1.0	0.6	37294	801	153	8.5	1.0	0.0	166605	2723	279	62	14	6.5
74	1.0	0.6	32611	1408	280	21.0	4.0	2.5	155762	6591	569	156	32	19.5
75	1.0	0.6	31697	878	100	14.0	3.0	1.0	153954	3892	338	104	40	16.0
76	0.2	0.8	0	3559	0	0.0	0.0	0.0	0	9044	0	0	0	0.0
77	0.2	0.8	0	1216	0	0.0	0.0	0.0	0	4693	0	0	0	0.0
78	0.2	0.8	0	4686	0	0.0	0.0	0.0	0	11079	0	0	0	0.0
79	0.2	0.8	0	3575	0	0.0	0.0	0.0	0	10034	0	0	0	0.0
80	0.2	0.8	0	3812	80	0.0	0.0	0.0	0	12717	0	0	0	0.0
81	0.4	0.8	404	7306	1780	157.5	15.0	6.0	819	29281	6336	421	7	0.0
82	0.4	0.8	338	9356	1678	29.0	0.0	0.0	358	20527	2229	172	10	8.5
83	0.4	0.8	247	8845	2507	47.0	2.5	0.0	597	33918	4657	452	133	130.0
84	0.4	0.8	81	5929	1168	0.0	0.0	0.0	204	19729	3098	2	0	0.0
85	0.4	0.8	175	9370	2011	16.5	0.0	0.0	336	28075	6811	152	2	0.0
86	0.6	0.8	8369	8680	2128	167.5	45.5	10.5	21426	36652	9575	843	264	238.5
87	0.6	0.8	10301	8076	1250	124.0	34.0	8.5	27079	42819	11190	980	168	64.0
88	0.6	0.8	8005	7424	1300	148.5	54.0	24.5	18204	32735	6807	967	162	79.0
89	0.6	0.8	7081	6877	1695	134.5	21.0	16.5	16880	26184	7043	836	61	38.5
90	0.6	0.8	9452	7779	2020	149.0	27.0	11.5	24959	39261	9533	947	102	24.5
91	0.8	0.8	21160	3814	616	63.5	11.0	7.0	70659	18792	4027	534	97	41.5
92	0.8	0.8	21750	4592	544	57.0	19.0	8.0	86487	29100	5434	382	126	57.0
93	0.8	0.8	20354	3313	417	46.5	17.0	8.0	89916	21807	3099	350	97	31.5
94	0.8	0.8	19522	3891	551	56.0	22.0	14.5	76612	20786	3926	508	125	41.0
95	0.8	0.8	25391	4376	490	60.0	15.0	9.0	88283	32830	6818	535	156	36.5
96	1.0	0.8	34166	778	140	9.0	0.0	0.0	133619	7526	584	186	47	28.0
97	1.0	0.8	34579	1321	288	16.0	6.0	3.0	170449	5263	882	119	30	17.0
98	1.0	0.8	41295	995	68	11.0	5.0	2.0	180642	4890	614	196	47	30.0
99	1.0	0.8	33049	1442	329	15.0	4.0	1.0	166670	5989	753	84	14	3.5

Table B.8(cntd) (a): $P4||\sum T_j$ (b): $P4||\sum w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
100	1.0	0.8	30714	2029	296	30.0	15.0	5.0	121005	9533	1529	334	78	33.5
101	0.2	1.0	0	2899	0	0.0	0.0	0.0	0	7972	0	0	0	0.0
102	0.2	1.0	0	3562	0	0.0	0.0	0.0	0	8840	0	0	0	0.0
103	0.2	1.0	0	3013	0	0.0	0.0	0.0	0	12007	0	0	0	0.0
104	0.2	1.0	0	3616	0	0.0	0.0	0.0	0	14168	124	0	0	0.0
105	0.2	1.0	0	3071	0	0.0	0.0	0.0	0	9035	0	0	0	0.0
106	0.4	1.0	0	6885	611	0.0	0.0	0.0	0	18781	538	0	0	0.0
107	0.4	1.0	325	12200	2410	42.0	5.5	2.0	639	33224	6814	362	24	5.5
108	0.4	1.0	0	9392	2169	0.0	0.0	0.0	0	29326	3999	30	0	0.0
109	0.4	1.0	66	9791	2261	93.0	4.0	2.5	292	33699	6038	305	5	0.5
110	0.4	1.0	28	9368	2330	6.0	0.0	0.0	56	38116	4895	308	0	0.0
111	0.6	1.0	14066	8586	1288	116.0	38.5	20.0	46851	52127	13876	898	130	41.0
112	0.6	1.0	13625	7846	1257	160.5	33.5	20.0	51161	40881	9117	905	192	80.0
113	0.6	1.0	9766	10753	1801	153.5	36.0	21.5	28933	50897	10803	912	196	61.0
114	0.6	1.0	13697	8641	1407	111.0	21.0	7.0	49881	40565	6329	828	137	45.0
115	0.6	1.0	7142	8559	2080	181.0	63.0	33.5	23131	44768	6136	963	163	76.5
116	0.8	1.0	24287	3719	518	49.5	13.5	5.5	101681	22605	3069	385	106	63.5
117	0.8	1.0	20649	4311	449	80.5	19.0	8.0	90672	27604	3565	552	138	42.5
118	0.8	1.0	21821	4821	541	82.0	19.0	4.5	68753	28214	3834	722	151	77.0
119	0.8	1.0	20853	4656	782	61.5	22.0	12.0	82257	27389	4490	548	162	67.5
120	0.8	1.0	21356	5053	841	63.5	21.5	10.0	74169	25293	5830	477	124	75.0
121	1.0	1.0	31140	2947	439	46.5	15.0	9.0	127667	17056	2917	392	83	37.0
122	1.0	1.0	36240	1616	277	8.0	4.0	2.5	153327	9180	430	114	34	11.0
123	1.0	1.0	27098	2739	497	35.0	10.5	6.0	108188	11216	2080	291	56	26.5
124	1.0	1.0	32096	2420	453	40.5	18.0	9.5	117039	16335	2181	264	80	44.5
125	1.0	1.0	34365	3103	597	52.0	20.0	6.0	150666	19827	3275	476	75	23.0

Table B.9 (a): $F20|\text{prmu}|\sum_{j=1}^{50} U_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{50} w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	12	18	8	3.0	1.0	1.0	23	64	30	14.0	3.5	3.0
2	0.2	0.2	8	26	10	3.0	1.0	0.0	16	64	49	12.5	2.5	1.0
3	0.2	0.2	5	32	8	3.0	1.0	0.0	10	111	42	11.0	4.5	2.0
4	0.2	0.2	4	38	4	2.0	0.5	0.0	10	74	37	15.5	5.0	2.5
5	0.2	0.2	5	42	7	1.5	1.0	0.0	21	96	28	12.5	5.0	1.0
6	0.4	0.2	24	17	6	2.0	1.0	1.0	104	72	28	11.5	1.5	1.0
7	0.4	0.2	20	25	9	2.5	1.0	0.0	54	89	38	12.5	4.0	2.5
8	0.4	0.2	16	32	9	3.0	1.0	0.0	51	88	43	17.0	4.0	2.5
9	0.4	0.2	14	34	6	3.0	1.0	0.5	76	93	63	17.5	6.0	6.0
10	0.4	0.2	15	31	9	3.0	1.0	1.0	63	105	38	11.0	6.0	0.0
11	0.6	0.2	36	12	5	2.0	0.0	0.0	173	41	41	8.0	2.0	1.0
12	0.6	0.2	32	14	9	2.0	1.0	1.0	124	58	26	5.0	4.0	2.0
13	0.6	0.2	28	22	9	2.0	1.0	0.0	140	83	44	11.0	4.5	3.0
14	0.6	0.2	24	26	10	3.0	1.0	1.0	147	122	55	12.5	3.0	2.0
15	0.6	0.2	23	27	6	3.0	1.0	1.0	111	65	20	6.5	0.0	0.0
16	0.8	0.2	48	2	0	0.0	0.0	0.0	269	9	6	0.0	0.0	0.0
17	0.8	0.2	44	6	1	1.0	0.0	0.0	215	37	37	1.5	0.0	0.0
18	0.8	0.2	40	10	1	1.0	0.0	0.0	214	57	33	1.5	0.0	0.0
19	0.8	0.2	35	15	3	2.0	1.0	1.0	178	99	19	6.5	0.0	0.0
20	0.8	0.2	32	18	4	1.0	0.0	0.0	127	42	19	3.5	1.0	0.0
21	1.0	0.2	50	0	0	0.0	0.0	0.0	274	0	0	0.0	0.0	0.0
22	1.0	0.2	50	0	0	0.0	0.0	0.0	251	0	0	0.0	0.0	0.0
23	1.0	0.2	49	0	0	0.0	0.0	0.0	280	9	0	0.0	0.0	0.0
24	1.0	0.2	45	5	1	1.0	0.0	0.0	247	35	3	0.0	0.0	0.0
25	1.0	0.2	42	8	0	0.0	0.0	0.0	217	47	0	0.0	0.0	0.0
26	0.2	0.4	10	22	8	4.0	1.5	1.0	16	62	40	7.5	2.0	0.5
27	0.2	0.4	6	27	8	3.0	1.0	1.0	11	62	46	9.5	3.0	1.5
28	0.2	0.4	4	33	9	2.0	1.0	0.0	7	75	29	12.5	3.0	3.0
29	0.2	0.4	4	39	9	2.0	1.0	1.0	13	81	35	15.5	4.0	2.0
30	0.2	0.4	7	32	3	2.0	1.0	1.0	18	41	17	8.0	3.5	1.5
31	0.4	0.4	22	19	7	2.0	1.0	0.0	77	76	23	14.5	4.0	1.5
32	0.4	0.4	17	28	9	4.0	1.0	1.0	53	103	49	14.0	6.0	4.0
33	0.4	0.4	15	31	6	4.0	1.0	1.0	54	96	39	15.0	6.0	2.5
34	0.4	0.4	15	34	12	2.0	1.0	0.0	68	72	37	7.0	1.0	1.0
35	0.4	0.4	16	34	7	1.0	0.5	0.0	90	75	41	2.0	1.0	1.0
36	0.6	0.4	33	8	5	2.0	0.0	0.0	179	83	54	14.0	4.0	4.0
37	0.6	0.4	28	14	10	2.0	1.0	1.0	132	80	22	7.0	2.0	0.0
38	0.6	0.4	25	15	10	2.5	1.0	1.0	116	104	41	11.0	2.5	0.0
39	0.6	0.4	22	20	12	2.0	1.0	1.0	125	110	58	8.5	2.5	2.0
40	0.6	0.4	21	21	10	2.5	1.0	1.0	137	98	25	5.0	1.0	1.0
41	0.8	0.4	45	1	1	0.0	0.0	0.0	225	28	7	0.0	0.0	0.0
42	0.8	0.4	41	4	3	1.0	0.0	0.0	215	42	0	0.0	0.0	0.0
43	0.8	0.4	38	10	3	1.0	0.0	0.0	200	68	0	0.0	0.0	0.0
44	0.8	0.4	34	11	8	1.0	0.0	0.0	199	87	59	5.0	0.0	0.0
45	0.8	0.4	31	13	7	2.0	0.0	0.0	180	78	34	6.0	0.0	0.0
46	1.0	0.4	50	0	0	0.0	0.0	0.0	271	0	0	0.0	0.0	0.0
47	1.0	0.4	50	0	0	0.0	0.0	0.0	276	0	0	0.0	0.0	0.0
48	1.0	0.4	46	2	1	0.0	0.0	0.0	229	21	0	0.0	0.0	0.0

Table B.9(cntd) (a): $F20|\text{prmu}|\sum_{j=1}^{50} U_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{50} w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
49	1.0	0.4	43	3	2	0.0	0.0	0.0	216	33	5	0.0	0.0	0.0
50	1.0	0.4	40	5	3	0.5	0.0	0.0	205	46	12	2.0	0.0	0.0
51	0.2	0.6	11	17	10	3.5	1.0	1.0	20	89	37	10.0	2.0	1.5
52	0.2	0.6	7	19	6	3.0	1.0	0.5	7	59	15	8.0	3.0	2.0
53	0.2	0.6	4	29	12	3.0	1.0	1.0	8	60	34	14.5	3.0	2.0
54	0.2	0.6	3	35	15	3.0	1.0	1.0	6	43	23	15.0	4.5	2.0
55	0.2	0.6	5	35	7	2.0	1.0	1.0	26	93	21	8.0	4.0	3.0
56	0.4	0.6	22	17	12	4.0	1.5	1.0	55	84	37	12.5	4.5	3.0
57	0.4	0.6	18	27	9	4.0	1.5	1.0	60	102	55	13.0	4.0	3.5
58	0.4	0.6	15	32	13	3.5	1.0	1.0	43	91	39	14.0	7.0	3.0
59	0.4	0.6	14	30	13	3.0	1.5	1.0	66	80	35	6.0	2.0	1.0
60	0.4	0.6	16	29	3	2.0	1.0	1.0	94	87	33	13.0	3.0	2.0
61	0.6	0.6	34	8	5	2.0	1.0	1.0	140	65	28	10.0	2.0	2.0
62	0.6	0.6	30	17	8	3.0	1.0	1.0	146	70	33	7.5	1.0	0.5
63	0.6	0.6	27	20	10	2.5	1.0	1.0	151	99	43	9.0	3.0	1.5
64	0.6	0.6	25	24	5	3.0	1.0	1.0	106	97	27	7.5	2.0	1.0
65	0.6	0.6	24	24	11	3.0	1.0	1.0	134	77	35	7.0	2.0	2.0
66	0.8	0.6	47	3	0	0.0	0.0	0.0	246	5	0	0.0	0.0	0.0
67	0.8	0.6	42	8	3	1.0	0.0	0.0	238	38	25	3.0	0.0	0.0
68	0.8	0.6	38	11	10	1.0	0.0	0.0	176	53	26	4.0	0.0	0.0
69	0.8	0.6	34	16	4	1.0	1.0	0.0	162	63	44	4.5	1.0	1.0
70	0.8	0.6	31	19	5	2.0	1.0	1.0	167	83	25	11.0	1.0	0.0
71	1.0	0.6	50	0	0	0.0	0.0	0.0	272	0	0	0.0	0.0	0.0
72	1.0	0.6	50	0	0	0.0	0.0	0.0	261	0	0	0.0	0.0	0.0
73	1.0	0.6	48	2	0	0.0	0.0	0.0	270	4	0	0.0	0.0	0.0
74	1.0	0.6	45	4	3	0.0	0.0	0.0	221	35	10	0.0	0.0	0.0
75	1.0	0.6	41	9	1	0.5	0.0	0.0	210	49	3	0.0	0.0	0.0
76	0.2	0.8	12	16	9	3.0	1.0	1.0	27	59	40	13.5	5.0	3.0
77	0.2	0.8	7	27	11	4.0	2.0	1.0	10	68	36	11.0	3.0	2.0
78	0.2	0.8	4	34	13	3.0	1.0	1.0	6	52	26	8.5	3.5	2.5
79	0.2	0.8	4	36	9	1.0	0.0	0.0	5	56	23	10.0	3.0	2.0
80	0.2	0.8	6	38	7	1.0	0.0	0.0	23	65	18	10.0	3.0	1.5
81	0.4	0.8	24	18	6	2.0	0.5	0.0	84	53	33	12.5	6.0	3.5
82	0.4	0.8	19	26	9	2.0	1.0	1.0	67	96	51	14.0	3.5	1.5
83	0.4	0.8	15	31	15	3.0	1.0	1.0	37	97	35	15.0	5.0	1.0
84	0.4	0.8	14	34	12	3.0	1.0	1.0	62	87	22	12.5	1.5	0.5
85	0.4	0.8	16	33	17	2.0	1.0	0.0	77	84	35	9.0	3.0	1.5
86	0.6	0.8	35	7	6	1.0	0.0	0.0	157	63	28	6.0	1.0	0.0
87	0.6	0.8	31	16	4	1.0	0.0	0.0	146	85	45	4.5	0.0	0.0
88	0.6	0.8	27	21	14	2.0	1.0	1.0	126	91	28	9.0	2.0	1.0
89	0.6	0.8	24	24	18	2.0	0.5	0.0	119	87	41	9.0	7.0	2.0
90	0.6	0.8	23	26	16	1.5	0.0	0.0	127	83	27	3.5	0.0	0.0
91	0.8	0.8	47	2	1	0.0	0.0	0.0	233	20	5	0.0	0.0	0.0
92	0.8	0.8	43	5	2	0.0	0.0	0.0	253	40	9	1.5	0.0	0.0
93	0.8	0.8	39	10	5	0.0	0.0	0.0	197	44	14	1.0	0.0	0.0
94	0.8	0.8	35	13	7	1.0	0.0	0.0	164	58	18	4.0	1.0	0.5
95	0.8	0.8	32	16	9	1.0	0.0	0.0	156	59	38	7.0	1.0	1.0
96	1.0	0.8	50	0	0	0.0	0.0	0.0	289	0	0	0.0	0.0	0.0
97	1.0	0.8	50	0	0	0.0	0.0	0.0	285	0	0	0.0	0.0	0.0
98	1.0	0.8	48	2	0	0.0	0.0	0.0	276	8	0	0.0	0.0	0.0
99	1.0	0.8	45	4	0	0.0	0.0	0.0	264	12	0	0.0	0.0	0.0
100	1.0	0.8	41	8	3	0.0	0.0	0.0	215	32	15	0.0	0.0	0.0
101	0.2	1.0	11	20	10	3.5	1.0	1.0	19	74	37	13.5	3.5	2.0
102	0.2	1.0	7	28	11	3.0	1.5	1.0	11	63	36	10.5	4.0	2.0
103	0.2	1.0	4	33	8	3.0	1.0	0.5	5	72	37	10.5	4.0	2.0
104	0.2	1.0	3	38	11	3.0	1.0	1.0	11	69	38	15.0	6.5	5.5
105	0.2	1.0	5	38	13	3.5	1.0	1.0	17	77	42	12.5	7.0	3.5
106	0.4	1.0	22	19	10	4.0	1.0	1.0	89	87	47	19.0	8.0	6.5
107	0.4	1.0	19	24	12	3.0	1.0	0.0	41	65	40	13.0	4.0	2.0
108	0.4	1.0	16	31	14	4.0	1.0	1.0	53	116	47	17.5	7.5	3.0
109	0.4	1.0	15	34	14	2.5	0.0	0.0	64	105	56	20.0	6.0	3.0
110	0.4	1.0	16	33	13	2.5	0.0	0.0	81	66	27	7.5	0.5	0.0
111	0.6	1.0	35	11	6	2.0	0.0	0.0	169	63	29	8.0	1.5	0.0
112	0.6	1.0	31	14	10	3.0	1.0	1.0	166	90	49	8.0	1.0	1.0
113	0.6	1.0	28	20	6	2.0	0.5	0.0	133	84	43	12.0	3.0	1.0
114	0.6	1.0	25	23	15	2.0	1.0	0.0	143	79	39	12.0	4.5	2.0
115	0.6	1.0	23	26	16	2.5	1.0	1.0	119	72	24	6.0	1.0	1.0
116	0.8	1.0	47	3	0	0.0	0.0	0.0	226	10	0	0.0	0.0	0.0
117	0.8	1.0	43	7	1	0.0	0.0	0.0	214	33	7	1.0	0.0	0.0
118	0.8	1.0	39	10	7	1.0	0.0	0.0	193	66	7	0.5	0.0	0.0
119	0.8	1.0	35	13	11	1.5	0.0	0.0	161	66	21	6.0	0.0	0.0
120	0.8	1.0	31	16	12	2.0	1.0	0.0	135	64	34	6.5	0.0	0.0
121	1.0	1.0	50	0	0	0.0	0.0	0.0	269	0	0	0.0	0.0	0.0
122	1.0	1.0	50	0	0	0.0	0.0	0.0	266	0	0	0.0	0.0	0.0
123	1.0	1.0	48	2	0	0.0	0.0	0.0	262	12	0	0.0	0.0	0.0
124	1.0	1.0	45	4	2	1.0	0.0	0.0	239	32	0	0.0	0.0	0.0
125	1.0	1.0	41	9	1	0.0	0.0	0.0	231	45	4	0.0	0.0	0.0

Table B.10 (a): $F20|\text{prmu}|\sum_{j=1}^{50} T_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{50} w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	5647	14735	7091	1540	4.0e+02	253	12618	57580	35151	6489	946	303
2	0.2	0.2	3072	16823	5592	1306	4.0e+02	272	9994	57317	23630	3087	586	440
3	0.2	0.2	1452	14958	6722	852	1.9e+02	45	5126	41574	22769	3247	332	122
4	0.2	0.2	1368	13551	4149	682	1.3e+02	87	3819	30810	19698	2263	389	192
5	0.2	0.2	4094	4930	2959	331	0.0e+00	0	16141	17099	7234	1536	586	366
6	0.4	0.2	24527	22660	6920	2826	7.9e+02	535	100883	94391	31484	13004	1619	214
7	0.4	0.2	23501	24217	7645	3008	9.1e+02	522	68431	76948	23595	5654	468	0
8	0.4	0.2	23339	23155	8155	2377	3.0e+02	78	77651	84116	20445	6580	455	194
9	0.4	0.2	24148	20364	6015	1738	2.3e+02	130	117843	76983	19003	7331	1847	332
10	0.4	0.2	25847	21280	6591	2372	2.3e+02	64	85237	71987	27879	4002	295	22
11	0.6	0.2	53256	28402	5572	2049	5.2e+02	192	225245	96791	22344	12251	2935	1827
12	0.6	0.2	54475	24504	4491	1846	5.1e+02	249	176935	85476	20113	4960	1219	734
13	0.6	0.2	55916	26270	7181	2698	5.9e+02	218	251349	62376	26570	4336	1358	751
14	0.6	0.2	57487	26039	7407	3204	5.9e+02	302	319473	115619	32004	10692	2451	1568
15	0.6	0.2	59371	23674	6230	2173	6.0e+02	238	226855	75734	17747	5216	476	285
16	0.8	0.2	89194	24775	4295	2618	9.3e+02	247	434762	91950	15963	7896	1871	769
17	0.8	0.2	90436	27052	7206	2726	9.8e+02	758	386360	86028	19620	9930	1355	330
18	0.8	0.2	92320	27809	6115	2312	5.2e+02	306	442339	79713	17900	8468	1009	234
19	0.8	0.2	93903	25747	4833	2380	6.4e+02	286	413804	61490	24640	6686	1388	921
20	0.8	0.2	999	117994	100492	97034	9.5e+04	94687	311727	49580	17757	6688	1336	653
21	1.0	0.2	999	149783	133745	129579	1.3e+05	126671	630539	85412	20391	12026	1535	438
22	1.0	0.2	999	151003	134965	131176	1.3e+05	127900	574359	75604	11248	6235	1366	258
23	1.0	0.2	999	153516	133036	131539	1.3e+05	129063	670213	88076	27429	9656	1646	548
24	1.0	0.2	999	156370	134930	132404	1.3e+05	130435	660989	66953	25516	8541	2652	484
25	1.0	0.2	999	161229	136158	133696	1.3e+05	131742	644225	77598	25457	8960	2336	486
26	0.2	0.4	999	16551	9710	5128	4.0e+03	3779	9588	30705	16778	5134	1300	551
27	0.2	0.4	999	14346	7492	2586	1.2e+03	1094	5213	30150	21639	13754	446	112
28	0.2	0.4	657	10997	3876	626	5.2e+01	23	2287	29875	13833	2134	198	70
29	0.2	0.4	999	8708	2767	892	6.4e+02	641	5563	29408	12010	1617	62	12
30	0.2	0.4	999	12050	5442	4328	3.9e+03	3920	11974	19263	5926	150	0	0
31	0.4	0.4	999	40329	29122	23999	2.2e+04	21862	70587	69955	23729	5998	1204	305
32	0.4	0.4	999	40732	27945	22966	2.1e+04	20968	60947	61637	20534	3318	851	456
33	0.4	0.4	999	41785	27584	22867	2.1e+04	21123	64564	60183	17204	3476	738	184
34	0.4	0.4	999	42974	28656	24149	2.3e+04	22698	78688	46586	14899	4885	527	26
35	0.4	0.4	999	47259	32376	26416	2.5e+04	24694	119505	62235	19793	5598	765	67
36	0.6	0.4	999	76638	56378	52648	5.0e+04	49728	241401	107500	32684	10052	2840	1210
37	0.6	0.4	999	74544	58222	53648	5.1e+04	50821	199634	93454	13920	8235	1510	553
38	0.6	0.4	52808	20147	6253	2941	6.8e+02	210	217837	72476	23666	9742	1390	303
39	0.6	0.4	54557	20263	7698	2801	5.8e+02	256	242971	89080	22121	11464	2730	704
40	0.6	0.4	56287	16173	7664	2108	4.9e+02	292	300579	83575	27066	9986	1500	242
41	0.8	0.4	84101	21972	4122	2154	6.5e+02	313	332735	76104	15508	4882	927	96
42	0.8	0.4	85705	21007	3829	2060	5.2e+02	260	391461	74110	14604	7358	1342	362
43	0.8	0.4	87151	22552	8189	2538	9.1e+02	262	432559	74970	22793	10712	2650	508
44	0.8	0.4	88878	27181	6383	3040	8.6e+02	520	474570	120000	30397	9394	2795	777
45	0.8	0.4	90804	24370	5818	3060	7.4e+02	362	431069	84290	23651	13026	956	71
46	1.0	0.4	120612	19536	4514	2114	4.2e+02	258	575828	66239	16869	6685	1912	1220
47	1.0	0.4	121714	19615	4593	2500	7.3e+02	306	599029	58843	16305	8509	1490	635
48	1.0	0.4	122990	20056	5895	2956	6.8e+02	376	561186	92075	15124	7116	1636	143
49	1.0	0.4	124689	24536	5559	2134	4.8e+02	270	577589	79436	16077	10637	2399	1550
50	1.0	0.4	126203	23355	4882	1996	8.9e+02	311	593490	102635	17751	6613	806	373
51	0.2	0.6	4801	12454	6110	2366	2.8e+02	98	10485	38226	19317	2799	724	318
52	0.2	0.6	2312	15087	8416	1431	3.6e+02	236	4453	29412	15761	2632	610	290
53	0.2	0.6	720	10796	6213	838	1.4e+02	99	1898	25344	14938	9085	332	185
54	0.2	0.6	778	8538	4522	2792	2.9e+01	11	3061	41002	17753	17753	352	248
55	0.2	0.6	999	15891	6593	2298	2.1e+03	2100	15270	48067	8076	1862	0	0
56	0.4	0.6	999	41053	29574	23603	2.2e+04	21326	53325	47505	19110	6058	1276	540
57	0.4	0.6	999	42740	29016	22198	2.1e+04	20458	64906	71485	24635	7196	972	236
58	0.4	0.6	999	47534	28360	22036	2.0e+04	20291	56888	56797	19731	3481	657	188
59	0.4	0.6	999	37542	26820	22986	2.1e+04	21118	67594	81304	18414	4952	835	208
60	0.4	0.6	999	37338	27375	24300	2.3e+04	22440	105551	105514	22014	7218	856	121
61	0.6	0.6	999	79155	52513	49958	4.8e+04	48097	166439	86613	30126	8442	1886	1498
62	0.6	0.6	49563	22253	6333	3428	6.9e+02	302	203559	71212	27895	10495	1560	890
63	0.6	0.6	51311	27843	7963	3726	8.6e+02	492	247299	122728	30257	11727	1606	396
64	0.6	0.6	53274	26155	8987	2142	4.8e+02	258	171059	46126	11679	3656	393	126
65	0.6	0.6	54987	26261	5615	2989	8.9e+02	414	256264	65292	23231	7335	665	76
66	0.8	0.6	82026	27516	4164	2154	5.2e+02	181	347231	97654	18846	10156	1598	441
67	0.8	0.6	82847	30886	6600	2227	1.0e+03	758	422639	94628	24994	9850	1596	946
68	0.8	0.6	84707	30344	5807	2024	7.0e+02	342	339357	86315	18374	11148	1239	556
69	0.8	0.6	86782	27959	6419	2445	4.3e+02	135	361199	96260	16384	7392	1594	188
70	0.8	0.6	88731	27167	9237	2594	4.0e+02	134	432801	93891	30948	15962	2721	2264
71	1.0	0.6	117914	26200	5888	2408	8.4e+02	555	562904	108117	18212	9916	602	195
72	1.0	0.6	119211	26061	5749	2114	5.2e+02	226	507760	87337	16331	7129	796	592
73	1.0	0.6	119932	26500	6188	2959	1.2e+03	788	593483	87530	15629	6221	1193	394
74	1.0	0.6	121276	27753	5124	1990	9.3e+02	499	532105	74950	23998	10165	1073	471
75	1.0	0.6	122706	31490	6085	2340	7.8e+02	478	584673	71231	24058	6749	2198	1242
76	0.2	0.8	5072	12881	6791	1789	4.3e+02	168	12415	40709	18097	4792	925	385
77	0.2	0.8	2469	13415	6854	1523	1.8e+02	94	4634	33801	20223	2898	570	306
78	0.2	0.8	707	11078	6397	6397	8.5e+01	0	2617	28972	19112	19112	824	81
79	0.2	0.8	1010	9487	4465	4465	3.0e+01	11	1890	28781	9547	9547	61	0
80	0.2	0.8	3672	6054	1762	505	2.3e+01	0	12542	39497	6761	320	0	0
81	0.4	0.8	23013	20282	7298	2924	6.9e+02	376	80694	81183	35058	9928	1175	590

Table B.10(cntd) (a): $F20|\text{prmu}|\sum_{j=1}^{50} T_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{50} w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
82	0.4	0.8	21983	17541	6280	2578	7.2e+02	382	72216	73409	24665	24665	766	224
83	0.4	0.8	22297	16571	6214	4830	4.7e+02	128	49176	84725	24820	5034	319	20
84	0.4	0.8	23604	16795	9034	9034	3.6e+02	153	82268	72619	26538	8844	227	31
85	0.4	0.8	25364	16924	4749	1946	3.4e+02	219	102022	95522	23434	7312	195	145
86	0.6	0.8	50449	22932	5692	2575	6.0e+02	194	202502	85328	19214	9390	750	480
87	0.6	0.8	51325	25021	6990	3332	7.7e+02	194	206132	70730	18745	4968	1518	545
88	0.6	0.8	52811	21044	7043	2951	7.0e+02	445	205231	72669	28132	9446	558	59
89	0.6	0.8	54655	22220	9040	3359	8.1e+02	435	229881	86747	35955	9270	1112	796
90	0.6	0.8	56619	21230	6797	2446	4.7e+02	192	223087	60733	14539	3717	264	246
91	0.8	0.8	85303	24116	5777	2698	1.0e+03	328	350440	73885	16148	5363	1166	300
92	0.8	0.8	86660	21788	6325	2724	7.1e+02	426	431891	113672	27535	11206	1590	495
93	0.8	0.8	88193	22115	6766	2750	5.1e+02	314	406699	97589	18190	10467	464	142
94	0.8	0.8	89338	24325	8209	3202	9.1e+02	541	382258	83868	25027	8162	1402	194
95	0.8	0.8	91037	25554	6305	2715	6.8e+02	419	410383	91755	20748	6638	618	369
96	1.0	0.8	122070	21471	5453	2426	7.9e+02	490	639788	94045	29333	15702	1110	441
97	1.0	0.8	123536	21183	5165	2000	5.0e+02	154	640719	67599	16236	7782	2100	1694
98	1.0	0.8	124397	21499	5481	2154	8.0e+02	558	645665	59939	28237	5270	1761	544
99	1.0	0.8	125797	23961	5849	3076	6.7e+02	359	651959	89641	28636	8664	3389	2045
100	1.0	0.8	127197	22631	5389	2424	4.9e+02	280	596138	60565	21710	7719	1833	836
101	0.2	1.0	4958	16532	5649	1404	3.2e+02	207	9234	44067	25060	4537	878	424
102	0.2	1.0	2518	11689	6165	1020	2.4e+02	129	4914	45187	19772	2352	484	242
103	0.2	1.0	933	15787	6245	6245	1.7e+02	78	2047	33022	16821	10498	610	120
104	0.2	1.0	882	12703	6536	3756	7.0e+00	0	3789	40776	16442	4566	688	84
105	0.2	1.0	3379	8128	1311	59	3.5e+00	0	9133	33519	6096	890	56	0
106	0.4	1.0	22469	23286	7357	2349	4.9e+02	207	85544	98102	24337	9730	2654	1505
107	0.4	1.0	21633	20895	6284	2034	4.1e+02	191	48696	52812	14937	4457	669	384
108	0.4	1.0	21981	24610	6522	2286	5.3e+02	285	65954	81503	27342	4735	156	17
109	0.4	1.0	23335	27656	7811	7811	5.4e+02	210	83711	76455	34226	5728	534	0
110	0.4	1.0	25124	25752	7251	1877	3.9e+02	234	102722	65551	18657	5051	356	158
111	0.6	1.0	50005	35168	4721	2834	5.3e+02	252	212837	83891	21923	6300	1645	544
112	0.6	1.0	51377	25936	7104	3024	4.7e+02	236	240536	82487	30199	10057	1550	618
113	0.6	1.0	53174	29771	6675	2751	5.7e+02	214	239714	84301	21209	7292	1071	708
114	0.6	1.0	54623	29930	6710	3110	1.0e+03	376	250680	90191	22682	5462	591	70
115	0.6	1.0	56510	29251	7090	2704	7.7e+02	522	249329	60312	20202	5565	613	316
116	0.8	1.0	84028	31086	4559	2150	8.0e+02	298	333603	76305	19150	4706	1246	188
117	0.8	1.0	85294	37046	5892	2881	6.0e+02	328	369841	72436	18509	6089	1894	834
118	0.8	1.0	86821	34291	6605	2635	7.6e+02	356	341352	82762	20784	5312	1040	40
119	0.8	1.0	88500	30044	6315	3302	6.5e+02	259	385270	111115	21282	4727	1247	398
120	0.8	1.0	89718	29187	7298	2764	6.5e+02	386	333659	69007	18766	6908	648	121
121	1.0	1.0	119821	31128	6727	2954	6.2e+02	331	582691	91523	18459	11004	1748	1082
122	1.0	1.0	121110	30987	6586	2861	4.6e+02	145	588191	84603	22183	8576	1152	396
123	1.0	1.0	122190	31057	4530	2512	7.4e+02	309	606761	105380	32029	8092	1430	1092
124	1.0	1.0	123147	34959	5559	3190	7.7e+02	425	622665	100043	17457	8870	792	3
125	1.0	1.0	124532	31319	5318	1744	6.4e+02	309	638057	88821	27792	12603	1526	320

Table B.11 (a): $F20|\text{prmu}|\sum_{j=1}^{100} U_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{100} w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	19	31	19	13.0	3.0	2.0	42	119	67	58.0	8.0	2.5
2	0.2	0.2	11	38	15	33.0	3.0	2.0	17	129	59	59.0	6.0	4.0
3	0.2	0.2	6	40	12	46.0	2.0	0.5	6	77	45	45.0	4.0	1.0
4	0.2	0.2	1	43	10	33.0	3.0	1.0	2	109	48	48.0	3.5	1.0
5	0.2	0.2	1	27	4	45.0	3.5	2.0	2	90	36	36.0	1.0	1.0
6	0.4	0.2	40	31	16	28.0	2.0	1.0	126	116	65	43.0	12.5	7.0
7	0.4	0.2	31	44	19	29.0	4.0	2.0	97	182	78	70.0	13.5	6.5
8	0.4	0.2	24	59	15	60.0	3.0	1.5	73	162	77	77.0	9.0	4.5
9	0.4	0.2	20	73	26	34.0	3.0	2.0	60	177	95	95.0	12.5	6.0
10	0.4	0.2	16	82	19	70.0	2.5	2.0	63	171	82	82.0	12.5	3.5
11	0.6	0.2	61	28	13	5.0	2.0	1.0	257	111	79	60.0	10.5	6.0
12	0.6	0.2	52	46	20	28.0	3.5	2.0	238	205	90	68.0	10.5	5.0
13	0.6	0.2	48	48	14	51.0	1.0	0.0	200	197	92	92.0	14.0	7.0
14	0.6	0.2	40	58	25	56.0	4.0	2.0	189	188	107	107.0	10.0	4.5
15	0.6	0.2	37	61	22	61.0	2.0	1.0	202	207	126	126.0	10.0	4.0
16	0.8	0.2	83	15	6	4.0	2.0	1.0	411	111	73	13.0	2.5	2.0
17	0.8	0.2	76	24	13	5.0	0.0	0.0	360	143	77	29.0	4.5	1.5
18	0.8	0.2	68	32	7	10.0	1.5	1.0	371	205	124	124.0	9.5	3.5
19	0.8	0.2	62	37	17	38.0	2.0	1.0	311	213	138	138.0	9.0	2.5
20	0.8	0.2	56	43	16	44.0	2.0	1.0	274	216	116	116.0	9.0	4.0
21	1.0	0.2	100	0	0	0.0	0.0	0.0	582	0	0	0.0	0.0	0.0
22	1.0	0.2	96	4	0	1.0	0.0	0.0	477	29	2	2.0	2.0	0.0
23	1.0	0.2	88	12	2	3.0	0.0	0.0	463	57	11	11.0	0.0	0.0
24	1.0	0.2	82	18	4	3.0	1.0	0.0	473	124	29	22.5	2.5	1.0
25	1.0	0.2	76	24	5	5.0	1.5	1.0	403	109	82	45.0	9.0	5.0
26	0.2	0.4	19	27	18	29.0	3.5	2.0	39	102	49	38.0	6.5	2.0
27	0.2	0.4	12	28	12	39.0	3.0	1.5	19	99	61	50.0	7.5	3.5
28	0.2	0.4	6	30	16	44.0	3.0	2.0	6	96	55	36.5	3.0	1.0
29	0.2	0.4	2	25	10	49.0	4.0	1.0	2	119	66	66.0	5.0	1.5

Table B.11(cntd) (a): $F20|\text{prmu}|\sum_{j=1}^{100} U_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{100} w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
30	0.2	0.4	2	12	1	45.0	4.0	1.5	2	99	48	48.0	6.0	2.5
31	0.4	0.4	41	30	14	32.0	2.5	1.0	106	125	98	98.0	7.5	4.0
32	0.4	0.4	32	43	16	44.0	3.0	2.0	95	164	101	101.0	9.0	4.0
33	0.4	0.4	26	54	19	59.0	3.0	1.0	68	161	91	64.0	13.0	6.0
34	0.4	0.4	21	64	24	73.0	3.0	1.0	70	202	103	103.0	15.0	3.0
35	0.4	0.4	18	79	29	80.0	2.0	1.0	80	181	105	105.0	11.5	7.0
36	0.6	0.4	63	18	10	25.0	2.0	1.0	217	107	71	53.0	12.0	7.5
37	0.6	0.4	55	28	20	43.0	1.0	0.5	228	151	82	64.0	12.5	5.0
38	0.6	0.4	47	45	23	52.0	3.0	2.0	183	174	130	130.0	14.5	6.0
39	0.6	0.4	42	51	30	58.0	3.0	1.0	217	235	160	160.0	12.0	5.0
40	0.6	0.4	37	57	27	47.0	4.0	2.0	224	180	131	131.0	11.0	1.5
41	0.8	0.4	84	12	7	5.0	1.0	1.0	430	114	34	30.0	4.5	0.0
42	0.8	0.4	76	19	9	7.0	1.5	1.0	392	120	71	15.0	5.5	2.0
43	0.8	0.4	68	24	16	10.0	2.0	1.0	336	175	109	25.0	4.5	1.5
44	0.8	0.4	61	32	19	39.0	2.0	1.0	320	183	138	36.5	6.5	3.0
45	0.8	0.4	57	35	25	33.0	1.5	0.0	291	165	108	108.0	9.5	4.0
46	1.0	0.4	100	0	0	0.0	0.0	0.0	581	0	0	0.0	0.0	0.0
47	1.0	0.4	96	2	1	1.0	0.0	0.0	520	18	4	4.0	0.0	0.0
48	1.0	0.4	88	10	6	2.0	1.0	1.0	526	76	37	23.0	1.0	0.0
49	1.0	0.4	82	15	6	6.0	0.0	0.0	455	121	73	34.0	3.5	2.0
50	1.0	0.4	75	20	13	5.5	1.0	0.0	382	136	95	26.0	6.0	2.5
51	0.2	0.6	20	25	12	25.0	2.0	0.5	36	79	53	35.0	6.0	1.5
52	0.2	0.6	11	31	12	39.0	3.0	2.0	13	79	39	30.5	5.5	2.0
53	0.2	0.6	5	38	12	46.0	2.5	0.5	5	68	25	23.0	4.0	1.0
54	0.2	0.6	0	39	14	25.0	1.0	0.0	1	64	39	31.0	2.0	1.0
55	0.2	0.6	0	21	3	4.0	0.0	0.0	0	59	31	24.5	6.0	1.0
56	0.4	0.6	40	27	11	30.0	3.0	1.0	125	123	68	35.0	6.0	2.5
57	0.4	0.6	31	42	12	43.0	3.0	1.5	106	193	123	123.0	8.0	2.5
58	0.4	0.6	24	57	14	57.0	2.0	1.0	57	187	76	76.0	10.0	5.0
59	0.4	0.6	20	76	11	60.0	3.0	1.0	54	203	81	81.0	13.0	3.5
60	0.4	0.6	18	82	17	65.0	2.0	1.0	82	198	100	100.0	10.5	3.5
61	0.6	0.6	62	22	12	28.0	2.0	1.0	252	145	66	44.0	7.5	4.5
62	0.6	0.6	53	33	14	43.0	2.0	2.0	191	188	92	92.0	15.0	7.0
63	0.6	0.6	45	48	14	36.0	3.0	2.0	202	197	98	98.0	10.5	4.5
64	0.6	0.6	40	56	23	42.0	2.0	1.0	197	204	112	112.0	9.0	5.0
65	0.6	0.6	35	62	24	45.0	3.0	2.0	157	189	112	112.0	6.0	2.0
66	0.8	0.6	84	11	9	4.0	1.0	0.0	369	77	24	14.0	5.0	1.0
67	0.8	0.6	75	19	9	8.0	2.0	1.0	408	98	63	24.0	5.0	3.5
68	0.8	0.6	67	25	14	7.0	2.0	1.0	352	150	71	31.0	7.5	2.0
69	0.8	0.6	60	32	21	32.0	1.0	1.0	308	168	100	46.0	9.5	4.0
70	0.8	0.6	55	38	22	31.0	2.0	1.0	271	165	73	73.0	7.5	3.5
71	1.0	0.6	100	0	0	0.0	0.0	0.0	514	0	0	0.0	0.0	0.0
72	1.0	0.6	98	1	0	0.0	0.0	0.0	534	13	2	2.0	0.0	0.0
73	1.0	0.6	89	10	3	2.0	0.0	0.0	442	48	22	15.0	0.0	0.0
74	1.0	0.6	80	17	9	6.0	1.0	0.5	442	105	34	27.0	5.0	0.0
75	1.0	0.6	73	22	14	7.0	2.0	1.0	382	122	83	40.0	3.5	1.0
76	0.2	0.8	20	24	14	25.0	2.0	0.0	26	80	60	43.0	8.0	3.0
77	0.2	0.8	11	33	13	10.5	2.0	1.0	15	64	29	25.0	5.5	2.0
78	0.2	0.8	4	32	12	20.0	2.0	1.0	6	65	29	24.5	3.0	0.5
79	0.2	0.8	0	23	7	6.0	1.0	1.0	1	63	37	24.5	5.5	1.0
80	0.2	0.8	0	0	0	0.0	0.0	0.0	0	66	21	18.0	5.0	1.5
81	0.4	0.8	40	29	11	9.0	3.0	1.0	127	136	56	43.0	11.0	6.5
82	0.4	0.8	30	45	15	10.0	4.0	2.0	63	141	95	41.5	8.0	3.0
83	0.4	0.8	24	57	27	56.0	2.0	1.0	65	194	114	114.0	10.5	2.5
84	0.4	0.8	19	68	13	63.0	3.0	2.0	79	236	117	117.0	15.0	4.0
85	0.4	0.8	17	81	13	76.0	1.5	1.0	69	159	67	67.0	11.0	4.5
86	0.6	0.8	62	20	9	29.0	2.5	1.0	255	107	57	52.5	9.0	3.0
87	0.6	0.8	53	29	16	41.0	2.0	1.0	203	170	83	60.0	11.0	5.0
88	0.6	0.8	46	40	20	36.0	2.0	1.0	168	164	88	88.0	8.0	3.0
89	0.6	0.8	41	41	28	41.0	2.5	1.0	174	187	117	117.0	12.5	7.0
90	0.6	0.8	37	49	16	41.0	4.0	1.0	185	194	106	106.0	10.0	5.5
91	0.8	0.8	84	9	8	4.0	1.0	1.0	441	69	49	16.0	2.0	1.0
92	0.8	0.8	76	15	13	13.5	1.0	1.0	375	113	77	17.0	4.0	1.0
93	0.8	0.8	68	25	17	29.0	1.5	1.0	335	153	90	60.5	6.5	3.0
94	0.8	0.8	61	30	21	37.0	2.0	1.0	322	170	66	40.5	8.5	4.0
95	0.8	0.8	55	37	19	31.0	2.5	1.0	262	159	67	44.0	7.5	4.0
96	1.0	0.8	100	0	0	0.0	0.0	0.0	543	0	0	0.0	0.0	0.0
97	1.0	0.8	95	4	2	1.0	0.0	0.0	532	29	5	5.0	0.0	0.0
98	1.0	0.8	88	11	3	2.0	0.0	0.0	456	59	21	21.0	0.0	0.0
99	1.0	0.8	81	18	5	4.0	1.0	0.5	451	101	82	18.0	2.0	0.0
100	1.0	0.8	74	24	12	5.0	2.0	1.0	361	103	66	15.0	4.0	1.5
101	0.2	1.0	18	24	17	13.0	4.5	3.0	46	109	67	56.0	6.0	3.0
102	0.2	1.0	11	29	13	30.0	2.0	1.0	10	74	35	28.0	4.0	2.0
103	0.2	1.0	5	28	10	32.0	2.0	0.5	4	82	48	37.0	4.0	2.0
104	0.2	1.0	0	21	4	30.0	3.0	1.0	1	73	26	20.0	3.0	0.0
105	0.2	1.0	0	0	0	34.0	1.0	0.0	0	55	24	19.5	3.0	1.0
106	0.4	1.0	40	26	13	31.0	2.5	1.5	102	116	67	44.0	7.5	3.5
107	0.4	1.0	32	39	14	31.0	2.0	1.0	74	157	91	91.0	10.0	3.0
108	0.4	1.0	24	54	23	37.0	4.0	2.0	55	196	83	83.0	12.0	6.5
109	0.4	1.0	20	63	26	36.0	2.5	1.5	52	172	67	67.0	13.0	4.5
110	0.4	1.0	17	75	31	47.0	5.0	2.0	81	217	115	115.0	16.5	6.0

Table B.11(cntd) (a): $F20|\text{prmu}|\sum_{j=1}^{100} U_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{100} w_j U_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
111	0.6	1.0	62	20	15	12.0	2.0	1.0	257	121	91	47.0	9.0	4.0
112	0.6	1.0	55	32	15	27.0	1.0	1.0	220	151	83	36.0	10.5	1.5
113	0.6	1.0	46	39	19	36.0	2.0	1.0	186	213	120	120.0	11.0	5.0
114	0.6	1.0	39	50	25	36.0	3.0	2.0	162	196	95	95.0	12.5	2.0
115	0.6	1.0	37	54	24	42.0	2.5	1.5	182	170	107	107.0	9.0	4.0
116	0.8	1.0	84	10	6	6.0	1.0	1.0	417	59	19	18.0	3.0	1.0
117	0.8	1.0	76	20	14	5.5	1.0	1.0	412	136	75	30.0	3.0	2.0
118	0.8	1.0	67	27	16	7.0	2.0	1.0	320	162	72	45.0	4.0	2.0
119	0.8	1.0	60	31	21	30.0	1.0	1.0	330	200	93	41.0	8.0	3.0
120	0.8	1.0	54	34	25	10.0	2.0	1.0	255	164	74	74.0	6.5	3.0
121	1.0	1.0	100	0	0	0.0	0.0	0.0	565	0	0	0.0	0.0	0.0
122	1.0	1.0	96	4	0	1.0	0.0	0.0	536	18	6	6.0	0.0	0.0
123	1.0	1.0	89	11	2	4.0	0.0	0.0	418	61	27	20.0	1.5	1.0
124	1.0	1.0	80	17	10	3.0	1.0	1.0	417	93	35	19.0	2.0	1.0
125	1.0	1.0	73	24	13	26.0	0.0	0.0	415	121	36	34.0	2.5	1.0

Table B.12 (a): $F20|\text{prmu}|\sum_{j=1}^{100} T_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{100} w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
1	0.2	0.2	12577	32825	16032	16032	1238	365	24364	147704	81897	81897	3536	1308
2	0.2	0.2	4241	28906	14640	14640	880	221	8416	116455	55407	55407	2425	1118
3	0.2	0.2	0	23984	11908	11908	18	0	18	61104	30313	30313	408	54
4	0.2	0.2	0	12169	3473	3473	0	0	0	41595	13952	13952	0	0
5	0.2	0.2	0	2436	342	0	0	0	0	11559	1014	507	0	0
6	0.4	0.2	61032	61915	19522	19522	1980	720	165345	172913	79774	79774	8627	2694
7	0.4	0.2	51032	61522	22997	22997	2466	688	137576	197793	93326	93326	9526	2306
8	0.4	0.2	42515	59842	24950	24950	2444	1039	101604	237562	91061	91061	6660	2206
9	0.4	0.2	36918	57289	23560	23560	3007	1438	80690	180164	56605	56605	5764	2376
10	0.4	0.2	33851	41944	21700	21700	2744	736	95670	174646	66184	66184	6260	1974
11	0.6	0.2	140771	81883	20079	20079	3270	1868	505470	230171	85659	85659	14762	8789
12	0.6	0.2	136533	83043	22871	22871	4598	1867	557750	257386	93545	93545	14342	5589
13	0.6	0.2	134856	55275	24744	24744	3578	906	508855	246075	79101	79101	7966	5166
14	0.6	0.2	134798	76610	25650	25650	3393	1394	495899	203093	64630	64630	8836	2820
15	0.6	0.2	135360	65064	20195	20195	3286	1696	590530	254615	86415	86415	11782	3835
16	0.8	0.2	247955	85968	13157	13157	2284	1002	1003001	191093	58795	58795	18108	8031
17	0.8	0.2	247456	82770	16322	16322	3626	2102	1010943	187533	61349	61349	10828	3994
18	0.8	0.2	248252	70723	14258	14258	3412	1288	1218390	279034	110762	110762	20814	9403
19	0.8	0.2	248450	87437	17930	17930	3074	1168	1121503	290426	65299	65299	14526	4296
20	0.8	0.2	248168	72898	22164	22164	3795	1554	1077298	287896	74104	74104	10904	6442
21	1.0	0.2	370724	82439	16838	16838	3258	1174	1839039	339678	62789	62789	22216	16144
22	1.0	0.2	370315	80576	15697	15697	1742	966	1474562	201798	45103	45103	12817	7476
23	1.0	0.2	369347	82153	16491	16491	3608	1613	1641349	201289	62976	62976	13067	4206
24	1.0	0.2	367837	86830	18003	18003	4124	2183	1883345	272078	72227	72227	12934	4918
25	1.0	0.2	368779	78098	20771	20771	3812	1540	1750002	222520	90453	90453	20110	8710
26	0.2	0.4	11841	38343	15205	15205	1027	415	30021	162510	79746	79746	2526	948
27	0.2	0.4	3795	32240	16450	16450	1026	346	10979	107675	65040	65040	2056	695
28	0.2	0.4	0	22813	10364	10364	0	0	0	53346	33833	33833	60	0
29	0.2	0.4	0	13332	3966	3966	0	0	0	33862	14110	14110	0	0
30	0.2	0.4	0	2287	102	0	0	0	0	8258	2099	2099	0	0
31	0.4	0.4	62390	58162	20483	20483	2480	1012	147839	203729	66188	66188	8110	2332
32	0.4	0.4	53113	63186	22086	22086	3410	1139	146045	189096	95206	95206	7368	2802
33	0.4	0.4	44691	58019	23916	23916	2642	1450	104728	192931	74651	74651	7796	3084
34	0.4	0.4	41110	51924	20640	20640	1748	320	97809	173721	57343	57343	5790	2571
35	0.4	0.4	38618	58526	25122	25122	3538	1463	124132	174287	55051	55051	5392	2608
36	0.6	0.4	146256	80497	17364	17364	2990	1162	425376	191138	65415	65415	12939	6169
37	0.6	0.4	144467	79858	19207	19207	4282	1130	522593	205952	74440	74440	11996	2682
38	0.6	0.4	143677	75485	19782	19782	4686	1466	464714	168482	77245	77245	11054	7714
39	0.6	0.4	143479	70609	23139	23139	3810	1198	630247	242576	78442	78442	17163	12692
40	0.6	0.4	144804	59787	21380	21380	3910	1173	657270	260956	86515	86515	13270	6494
41	0.8	0.4	254359	89358	14615	14615	3085	1589	1074097	366574	67785	67785	16136	9488
42	0.8	0.4	256024	91983	17176	17176	3373	1241	1199529	297401	98400	98400	16238	8382
43	0.8	0.4	257798	82111	15838	15838	3136	1240	1101284	267986	79333	79333	10973	4795
44	0.8	0.4	258308	77324	17097	17097	3050	1230	1201376	294667	79798	79798	16648	5818
45	0.8	0.4	258971	78375	16027	16027	2950	1438	1149600	274441	93897	93897	11154	3400
46	1.0	0.4	377178	101111	15184	15184	2263	469	1867341	367632	71651	71651	12546	5380
47	1.0	0.4	375504	103596	18786	18786	3570	1966	1735211	257675	69366	69366	14652	7822
48	1.0	0.4	375676	96409	16833	16833	4466	1972	1997368	351105	104608	104608	13980	5600
49	1.0	0.4	376050	88804	14770	14770	2666	1090	1901460	319839	103494	103494	22750	10480
50	1.0	0.4	376594	78714	15240	15240	3908	2152	1734300	274429	87497	87497	13875	4115
51	0.2	0.6	10734	34319	15735	15735	1423	422	21321	67203	35041	35041	3221	1752
52	0.2	0.6	2859	28380	13773	13773	669	331	7488	64300	32485	32485	1814	361
53	0.2	0.6	0	20438	8887	8887	0	0	0	69936	32030	32030	0	0
54	0.2	0.6	0	6922	997	0	0	0	0	20555	6668	6668	0	0
55	0.2	0.6	0	0	0	0	0	0	0	17550	2743	2743	0	0
56	0.4	0.6	58884	53556	19960	19960	1908	462	172547	158508	61818	61818	7327	1256
57	0.4	0.6	49299	48858	20265	20265	1670	628	140716	177900	78165	78165	11359	3110
58	0.4	0.6	40003	59364	23043	23043	2760	1432	80998	145409	75596	75596	6781	3778

Table B.12(cntd) (a): $F20|\text{prmu}|\sum_{j=1}^{100} T_j$ (b): $F20|\text{prmu}|\sum_{j=1}^{100} w_j T_j$

	TF	RDD	bk	AU	NEH	ILS.1	ILS.60	ILS.300	bk	AU	NEH	ILS.1	ILS.60	ILS.300
59	0.4	0.6	34099	47383	19527	19527	2273	924	76344	157649	49029	49029	5692	2336
60	0.4	0.6	31852	61544	26312	26312	3057	1134	104106	204444	66583	66583	5496	3124
61	0.6	0.6	139975	73503	21076	21076	4164	1970	491080	222150	52097	52097	11618	2942
62	0.6	0.6	137214	74262	18648	18648	3076	1242	457264	192579	77364	77364	7612	3083
63	0.6	0.6	135766	76479	19743	19743	3304	1539	548913	229363	66799	66799	12722	6000
64	0.6	0.6	135862	80372	18118	18118	3261	1082	522900	237391	74548	74548	16879	3730
65	0.6	0.6	135987	72761	21974	21974	4642	1096	448826	200627	63502	63502	8641	4050
66	0.8	0.6	250962	88067	15615	15615	2825	1392	857926	191347	58118	58118	13224	5648
67	0.8	0.6	250633	80644	18857	18857	4314	2298	1175340	366418	80213	80213	13724	7053
68	0.8	0.6	252990	85099	18495	18495	2350	496	1199978	288078	70900	70900	14942	7470
69	0.8	0.6	252695	85398	16585	16585	3570	1342	1159747	385759	62531	62531	13536	6574
70	0.8	0.6	252750	83704	21235	21235	5034	1601	1118271	350775	82975	82975	15850	3919
71	1.0	0.6	374053	90604	13761	13761	3150	1545	1660770	255549	73481	73481	15291	3684
72	1.0	0.6	373500	91153	15803	15803	2964	1533	1774399	291959	58714	58714	15460	7948
73	1.0	0.6	372517	79347	17044	17044	3778	1606	1627164	243214	58447	58447	11142	3654
74	1.0	0.6	372548	91255	16524	16524	4258	2192	1867309	345109	93025	93025	15194	6838
75	1.0	0.6	999	472897	387163	387163	375911	373885	1717077	268211	80638	80638	13435	7745
76	0.2	0.8	999	49704	26693	26693	12152	11572	19340	69155	39490	39490	1328	326
77	0.2	0.8	999	32680	18879	18879	4127	3676	7969	63878	33197	33197	1859	1194
78	0.2	0.8	0	23468	11099	11099	54	15	147	43023	19807	19807	77	28
79	0.2	0.8	0	9371	3217	2108	0	0	0	22797	10816	10816	0	0
80	0.2	0.8	0	0	0	0	0	0	0	15617	1149	1149	0	0
81	0.4	0.8	999	116880	79389	79389	61056	60130	185652	199629	79010	79010	5330	1734
82	0.4	0.8	999	102814	70719	70719	50961	49642	94560	129503	63947	63947	2748	1180
83	0.4	0.8	999	100793	59449	59449	42146	41072	92213	186811	77448	77448	6205	2601
84	0.4	0.8	999	84920	54292	54292	36482	36035	113169	247887	99620	99620	6251	2822
85	0.4	0.8	999	87117	53955	53955	34242	33521	98512	156481	64875	64875	6382	1242
86	0.6	0.8	999	208281	158391	158391	144710	141572	505396	217951	84228	84228	13584	4620
87	0.6	0.8	999	217700	156269	156269	140807	138460	455418	219009	80104	80104	11146	5631
88	0.6	0.8	999	217967	157116	157116	137456	135353	401468	144368	60694	60694	6280	2382
89	0.6	0.8	999	205796	154837	154837	138190	136418	470386	223271	94758	94758	11128	6692
90	0.6	0.8	999	206000	155340	155340	138816	137515	541584	193121	81568	81568	12226	3424
91	0.8	0.8	999	341372	263251	263251	254206	252708	1051187	315638	67748	67748	12956	6214
92	0.8	0.8	999	334224	270717	270717	254077	252828	1083326	283247	70609	70609	10560	2866
93	0.8	0.8	999	343760	268582	268582	254658	253743	1091837	261664	80987	80987	12378	5230
94	0.8	0.8	999	336144	273615	273615	255310	253078	1183824	315751	99666	99666	13385	6131
95	0.8	0.8	999	343869	274146	274146	255854	253546	1054307	313631	95700	95700	15776	5876
96	1.0	0.8	999	460166	394143	394143	377784	376966	1663355	269693	63786	63786	12052	6470
97	1.0	0.8	999	471984	391955	391955	377958	375878	1763545	282681	68578	68578	12986	8014
98	1.0	0.8	999	462127	388070	388070	377647	376292	1743638	289753	56971	56971	12350	3369
99	1.0	0.8	375163	91493	15456	15456	2934	1696	1862709	272482	66618	66618	14794	8282
100	1.0	0.8	375279	87511	16808	16808	4015	1944	1621680	195347	68344	68344	11776	4413
101	0.2	1.0	10980	36276	15653	15653	1242	209	29691	109383	56998	56998	2618	506
102	0.2	1.0	2721	29562	15256	15256	917	331	6110	52817	26095	26095	1538	848
103	0.2	1.0	0	16989	10214	10214	0	0	0	46746	25665	25665	0	0
104	0.2	1.0	0	4371	776	0	0	0	0	14234	5461	5461	0	0
105	0.2	1.0	0	0	0	0	0	0	0	18059	395	395	0	0
106	0.4	1.0	59992	60540	19546	19546	2513	565	133284	136433	72603	72603	8932	4398
107	0.4	1.0	48866	58015	24123	24123	2780	590	102670	191021	77602	77602	6464	2346
108	0.4	1.0	37855	59644	22479	22479	3427	2384	68605	215361	65040	65040	8930	3832
109	0.4	1.0	30851	45950	30018	30018	2960	1286	62286	196328	79611	79611	5340	3197
110	0.4	1.0	29471	44885	22159	22159	2888	1160	115357	211274	66265	66265	6832	3343
111	0.6	1.0	140055	82841	19357	19357	3604	1476	480837	248034	70022	70022	8002	3836
112	0.6	1.0	137610	75618	23115	23115	4243	1518	464471	225798	71791	71791	13482	5893
113	0.6	1.0	137190	54360	23133	23133	3010	1251	469561	229862	78641	78641	14494	5114
114	0.6	1.0	137437	68967	20385	20385	3490	666	455904	191260	61020	61020	13170	5774
115	0.6	1.0	137778	77291	24976	24976	4758	945	513795	214183	60695	60695	13188	6239
116	0.8	1.0	248690	87224	18292	18292	3175	1046	1037654	290119	62589	62589	17630	10036
117	0.8	1.0	249967	92552	21529	21529	3544	1168	1164264	266522	77233	77233	26950	11350
118	0.8	1.0	250877	79394	19041	19041	3539	1750	999141	263332	60746	60746	14976	7850
119	0.8	1.0	252707	78713	17732	17732	3226	786	1204693	274221	92848	92848	15926	7413
120	0.8	1.0	253713	90625	21197	21197	3058	754	991139	236829	63380	63380	11718	5112
121	1.0	1.0	372823	93443	14007	14007	3040	1978	1776747	227513	53395	53395	15858	4370
122	1.0	1.0	372407	93270	14104	14104	2544	1335	1818517	309801	71592	71592	16576	4260
123	1.0	1.0	371603	96541	16780	16780	3047	1364	1499642	189220	80317	80317	12546	4208
124	1.0	1.0	371713	90402	16027	16027	4054	1858	1767845	281338	62959	62959	14759	4344
125	1.0	1.0	373028	98108	17605	17605	2559	894	1904157	200787	73989	73989	18244	10084

Bibliography

[Aart 97] E. H. L. Aarts and J. K. Lenstra, Eds. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.

[Aden 01] B. Adenso-Díaz and M. Laguna. "Automated Fine-tuning of Algorithms with Taguchi Fractional Experimental Designs and Local Search". Tech. Rep., Graduate School of Business and Administration, University of Colorado at Boulder, CO, USA, 2001.

[Aden 92] B. Adenso-Diaz. "Restricted Neighborhood in the Tabu Search for the Flow Shop Problem". *European Journal of Operational Research*, Vol. 62, No. 1, pp. 27–37, 1992.

[Aika 74] H. Aikake. "A New Look at Statistical Model Identification". *IEEE Transactions on Automatic Control*, Vol. 19, pp. 716–723, 1974.

[Ande 82] P. Andersen and R. Gill. "Cox's Regression Model for Counting Processes: A Large Sample Study". *Annals of Statistics*, Vol. 10, pp. 1100–1120, 1982.

[Ande 97] E. J. Anderson, C. A. Glass, and C. N. Potts. "Machine scheduling". In: E. H. L. Aarts and J. K. Lenstra, Eds., *Local Search in Combinatorial Optimization*, Chap. 11, pp. 361–414, John Wiley & Sons, Chichester, UK, 1997.

[Aziz 98] M. Azizoglu and O. Kirca. "Tardiness Minimization on Parallel Machines". *International Journal of Production Economics*, Vol. 55, No. 2, pp. 163–168, 1998.

[Bake 73] K. R. Baker. "Procedures for Sequencing Tasks With One Resource Type". *International Journal of Production Research*, Vol. 11, pp. 125–138, 1973.

[Bake 77] K. R. Baker. "Computational Experience With a Sequencing Algorithm Adapted to the Tardiness Problem". *AIIE Transactions*, Vol. 9, pp. 32–35, 1977.

[Bake 82] K. R. Baker and J. W. M. Bertrand. "A Dynamic Priority Rule for Scheduling Against Due-Dates". *Journal of Operations Management*, Vol. 1, No. 1, pp. 37–42, 1982.

[Barb 00] L. Barbalescu, J.-P. Watson, and L. D. Whitley. "Dynamic Representations and Escaping Local Optima: Improving Genetic Algorithms and Local Search". In: *Proceedings of AAAI'2000*, 2000. Available from http://www.cs.colostate.edu/~genitor/Pubs.html.

[Barr 96] R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart. "Designing and Reporting on Computational Experiments with Heuristic Methods". *Journal of Heuristics*, Vol. 1, No. 1, pp. 9–32, 1996.

[Baue 00] A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. "Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization". *Central European Journal for Operations Research*, Vol. 8, No. 2, pp. 125–141, 2000.

[Bean 94] J. C. Bean. "Genetic Algorithms and Random Keys for Sequencing and Optimization". *ORSA Journal on Computing*, Vol. 6, pp. 154–160, 1994.

[Beas 03] J. E. Beasley. "OR-Library". Available from http://mscmga.ms.ic.ac.uk/info.html, 2003. Version visited last on 15 September 2003.

[Beas 90] J. E. Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". *Journal of the Operational Research Society*, Vol. 41, No. 11, pp. 1069–1072, 1990.

[Beck 97] J. C. Beck, A. J. Davenport, and M. S. Fox. "Five Pitfalls of Empirical Scheduling Research". In: *Proceedings of the Third International Conference on Principles and Practices of Constraint Programming*, pp. 390–404, 1997.

[Best 00a] M. L. den Besten. *Ants for the Single Machine Total Weighted Tardiness Scheduling Problem*. Master's thesis, Universiteit van Amsterdam, Amsterdam, NL, April 2000.

[Best 00b] M. den Besten, T. Stützle, and M. Dorigo. "Ant Colony Optimization for the Total Weighted Tardiness Problem". In: G. Rudolph, X. Yao, E. Luton, J. J. Merelo, and H.-P. Schwefel, Eds., *Proceedings of PPSN-VI*, pp. 611–620, Springer Verlag, Berlin, DE, September 2000.

[Best 01a] M. den Besten and T. Stützle. "Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling". In: *Proceedings of MIC'2001 — 4th Metaheuristics International Conference*, pp. 545–555, Porto, PT, 2001.

[Best 01b] M. den Besten, T. Stützle, and M. Dorigo. "Design of Iterated Local Search Algorithms: An Example Application to the Single Machine Total Weighted Tardiness Problem". In: E. J. W. Boers, J. Gottlieb, P. L.

Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, Eds., *Applications of Evolutionary Computing*, pp. 441–451, Springer Verlag, Berlin, DE, April 2001.

[Bilg 04] U. Bilge, F. Kıraç, M. Kurtulan, and P. Pekgün. "A Tabu Search Algorithm for Parallel Machine Total Tardiness Problem". *Computers & Operations Research*, Vol. 31, No. 3, pp. 397–414, March 2004.

[Bira 02] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. "A Racing Algorithm for Configuring Metaheuristics". In: W. B. Langdon *et al.*, Eds., *GECCO 2002: Genetic and Evolutionary Computation Conference*, pp. 11–18, Morgan Kaufmann, San Francisco, CA, USA, 2002.

[Bira 03] M. Birattari. "The *Race* Package for R: Racing Methods for the Selection of the Best". Tech. Rep. 37, IRIDIA, ULB, Brussels, BE, November 2003. Package Available from http://cran.r-project.org/src/contrib/Descriptions/race.html.

[Bira 04] M. Birattari. *The Problem of Tuning Metaheuristics as seen from a Machine Learning Perspective*. PhD thesis, ULB, Brussels, BE, 2004.

[Bohm 66] C. Böhm and G. Jacopini. "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". *Communications of ACM*, Vol. 9, No. 5, pp. 366–371, 1966.

[Bona 99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, USA, 1999.

[Brad 85] R. M. Brady. "Optimization Strategies Gleaned from Biological Evolution". *Nature*, Vol. 317, pp. 804–806, 1985.

[Brei 84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, USA, 1984.

[Broo 95] F. P. Brooks. *The Mythical Man–Month*. Addison-Wesley, Reading, MA, USA, anniversary Ed., 1995.

[Bruc 03] P. Brucker and S. Knust. "Complexity Results for Scheduling Problems". Available from http://www.mathematik.uni-osnabrueck.de/research/OR/class, 2003. Version visited last on 15 September 2003.

[Carr 65] D. C. Carroll. *Heuristic Sequencing of Single and Multiple Components*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1965.

[Chan 90] S. Chang, H. Mathuo, and G. Tang. "Worst-Case Analysis of Local Search Heuristics for the One-Machine Total Tardiness Problem". *Naval Research Logistics*, Vol. 37, pp. 111–121, 1990.

[Chia 03a] M. Chiarandini, K. Socha, M. Birattari, and O. Rossi-Doria. "An effective Hybrid Approach for the University Course Timetabling Problem". 2003. submitted to *Journal of Scheduling*.

[Chia 03b] M. Chiarandini, K. Socha, M. Birattari, and O. Rossi-Doria. "International Timetabling Competition. A Hybrid Approach". Tech. Rep. AIDA–03–04, Darmstadt University of Technology, Department of Computer Science, Darmstadt, DE, March 2003.

[Cho 81] Y. Cho and S. Sahni. "Preemptive Scheduling of Independent Jobs with Release and Due Times on Open, Flow and Job Shops". *Operations Research*, Vol. 29, pp. 511–522, 1981.

[Cohe 95] P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1995.

[Coll 88] N. E. Collins, R. W. Eglese, and B. L. Golden. "Simulated Annealing — An Annotated Bibliography". *American Journal of Mathematical and Management Sciences*, Vol. 8, No. 3-4, pp. 209–307, 1988.

[Cong 02] R. K. Congram, C. N. Potts, and S. van de Velde. "An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem". *INFORMS Journal on Computing*, Vol. 14, No. 1, pp. 52–67, 2002.

[Cono 99] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, third Ed., 1999.

[Conw 67] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading, MA, USA, 1967.

[Cook 97] W. J. Cook, W. H. Cunningham, W. R. Pulleybank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, New York, NY, USA, 1997.

[Coy 00] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. "Using Experimental Design to Find Effective Parameter Settings for Heuristics". *Journal of Heuristics*, Vol. 7, pp. 77–97, 2000.

[Crau 98] H. A. J. Crauwels, C. N. Potts, and L. N. van Wassenhove. "Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem". *INFORMS Journal on Computing*, Vol. 10, No. 3, pp. 341–350, 1998.

[Croc 98] F. D. Croce, R. Tadei, P. Baracco, and A. Grosso. "A New Decomposition Approach for the Single Machine Total Tardiness Scheduling Problem". *Journal of the Operational Research Society*, Vol. 49, pp. 1101–1106, 1998.

[Crow 79] H. Crowder, R. Dembo, and J. Mulvey. "On Reporting Computational Experiments with Mathematical Software". *ACM Transactions on Mathematical Software*, Vol. 5, No. 2, pp. 193–203, 1979.

[Culb 96] J. C. Culberson. "On the Futility of Blind Search". Tech. Rep. 18, Department of Computing Science, University of Alberta, Edmonton, CA, July 1996.

[Dean 99] A. Dean and D. Voss. *Design and Analysis of Experiments*. Springer Verlag, New York, NY, USA, 1999.

[DeJo 90] K. A. DeJong and W. M. Spears. "An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms". In: *Proceedings of the First Workshop on Parallel Problem Solving from Nature*, pp. 38–47, Springer Verlag, Berlin, DE, 1990.

[Dogr 79] A. Dogramaci and J. Surkis. "Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors". *Management Science*, Vol. 25, pp. 1208–1216, 1979.

[Dori 04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004.

[Dori 99] M. Dorigo and G. Di Caro. "The Ant Colony Optimization Meta-Heuristic". In: D. Corne, M. Dorigo, and F. Glover, Eds., *New Ideas in Optimization*, pp. 11–32, McGraw Hill, London, UK, 1999.

[Du 90] J. Du and J. Y.-T. Leung. "Minimizing Total Tardiness on One Machine is NP–Hard". *Mathematics of Operations Research*, Vol. 15, No. 3, pp. 483–495, 1990.

[Elma 68] S. E. Elmaghraby. "The One Machine Sequencing Problem with Delay Costs". *Journal of Industrial Engineering*, Vol. 19, pp. 105–108, 1968.

[Erle 01] T. Erlebach, H. Kellerer, and U. Pferschy. "Approximating Multi–Objective Knapsack Problems". In: F. K. H. A. Dehne, J.-R. Sack, and R. Tamassia, Eds., *Proceedings of the Seventh International Workshop on Algorithms and Data Structures*, pp. 210–221, Springer Verlag, Berlin, DE, 2001.

[Ever 01] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Oxford University Press, New York, NY, USA, 2001.

[Fara 02] J. J. Faraway. *Practical Regression and Anova using R*. July 2002. Available from http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf.

[Fish 70] P. C. Fishburn. *Utility Theory for Decision Making*. John Wiley & Sons, New York, NY, USA, New York, NY, USA, 1970.

[Fry 89] T. D. Fry, L. Vicens, K. Macleod, and S. Fernandez. "A Heuristic Solution Procedure to Minimize $\overline{T}$ on a Single Machine". *Journal of the Operational Research Society*, Vol. 40, pp. 293–297, 1989.

[Gare 78] M. R. Garey and D. S. Johnson. "Strong NP-completeness results: Motivation, Examples and Implications". *Journal of the Association for Computing Machinery*, Vol. 25, pp. 499–508, 1978.

[Gare 79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, USA, 1979.

[Gent 97] I. P. Gent, S. A. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. M. Smith, and T. Walsh. "How Not To Do It". Research Report 27, School of Computing Research, University of Leeds, UK, May 1997.

[Glov 89] F. Glover. "Tabu Search – Part I". *ORSA Journal on Computing*, Vol. 1, No. 3, pp. 190–206, 1989.

[Glov 90] F. Glover. "Tabu Search – Part II". *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 4–32, 1990.

[Gold 85] B. L. Golden and W. Steward. "Empirical Analysis of Heuristics". In: E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, Eds., *The Traveling Salesman Problem*, pp. 207–249, John Wiley & Sons, Chichester, UK, 1985.

[Gold 89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, USA, 1989.

[Gonz 78] T. Gonzalez and S. Sahni. "Flow Shop and Job Shop Schedules: Complexity and Approximation". *Operations Research*, Vol. 26, pp. 36–52, 1978.

[Good 01] P. I. Good. *Resampling Methods*. Birkhauser, Boston, MA, USA, second Ed., 2001.

[Grah 79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey". In: P. L. Hammer, E. L. Johnson, and B. H. Korte, Eds., *Discrete Optimization*, pp. 287–326, North-Holland, Amsterdam, NL, 1979.

[Gros 04] A. Grosso, F. Della Croce, and R. Tadei. "An Enhanced Dynasearch Neighborhood for the Single-Machine Total Weighted Tardiness Scheduling Problem". *Operations Research Letters*, Vol. 32, No. 1, pp. 68–72, 2004.

[Gupt 73] J. N. D. Gupta and A. R. Maykut. "Scheduling Jobs on Parallel Processors With Dynamic Programming". *Decision Science*, Vol. 4, pp. 447–457, 1973.

[Hans 03] P. Hansen and N. Mladenović. "A Tutorial on Variable Neighborhood Search". Les cahiers du GERAD 46, GERAD, Montreal, CA, July 2003.

[Hans 99a] P. Hansen and N. Mladenović. "An Introduction to Variable Neighborhood Search". In: S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Eds., *Meta-heuristics: Advances and trends in local searchs paradigms for optimization*, pp. 433–458, Kluwer Academic Publishers, Dordrecht, NL, 1999.

[Hans 99b] P. Hansen and N. Mladenović. "Variable Neighborhood Search: Methods and Recent Applications". In: *Proceedings of MIC'99*, pp. 275–280, 1999.

[Hark 87] P. T. Harker. "Incomplete Pairwise Comparisons in the Analytic Hierarchy Process". *Mathematical Modelling*, Vol. 9, pp. 837–848, 1987.

[Haup 89] R. Haupt. "A Survey of Priority Rule-Based Scheduling". *OR Spektrum*, Vol. 11, pp. 3–16, 1989.

[Ho 91] J. C. Ho and Y. L. Chang. "Heuristics for Minimizing Mean Tardiness for m Parallel Machines". *Naval Research Logistics*, Vol. 38, pp. 367–381, 1991.

[Hoet 99] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. "Bayesian Model Averaging: A Tutorial". *Statistical Science*, Vol. 14, No. 4, 1999.

[Hols 92] J. E. Holsenback and R. M. Russell. "A Heuristic Algorithm for Sequencing on One Machine to Minimize Total Tardiness". *Journal of the Operational Research Society*, Vol. 43, pp. 53–62, 1992.

[Hook 94] J. N. Hooker. "Needed: An Empirical Science of Algorithms". *Operations Research*, Vol. 42, No. 2, pp. 201–212, 1994.

[Hook 96] J. N. Hooker. "Testing heuristics: We have it all wrong". *Journal of Heuristics*, Vol. 1, No. 1, pp. 33–42, 1996.

[Hoos 04] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, USA, August 2004.

[Jack 55] J. R. Jackson. "Scheduling a Production Line to Minimize Maximum Tardiness". Research Report 43, Management Science Research Project, UC LA, Los Angeles, CA, USA, 1955.

[John 02] D. S. Johnson. "A Theoretician's Guide to the Experimental Analysis of Algorithms". In: M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, Eds., *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pp. 215–250, American Mathematical Society, 2002.

[Karp 72] R. M. Karp. "Reducibility Among Combinatorial Problems". In: R. E. Miller and J. W. Thatcher, Eds., *Complexity of Computer Computations*, pp. 85–103, Plenum Press, New York, NY, USA, 1972.

[Kenn 00] P. Kennedy. *A Guide to Econometrics*. Blackwell Publishers, Oxford, UK, fourth Ed., 2000.

[Kim 93a] Y. D. Kim. "Heuristics for Flowshop Scheduling Problems Minimizing Mean Tardiness". *Journal of the Operational Research Society*, Vol. 44, pp. 19–28, 1993.

[Kim 93b] Y.-D. Kim. "A New Branch and Bound Algorithm for Minimizing Mean Tardiness in Two-Machine Flowshops". *Computers and Operations Research*, Vol. 20, No. 4, pp. 391–401, May 1993.

[Kirk 83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. "Optimization by Simulated Annealing". *Science*, Vol. 220, pp. 671–680, 1983.

[Koul 93] C. P. Koulamas. "An Effective Heuristic for the Parallel Machine Tardiness Problem". Tech. Rep., Florida International University, Miami, FL, USA, 1993.

[Koul 94] C. P. Koulamas. "The Total Tardiness Problem: Review and Extensions". *Operations Research*, Vol. 42, No. 6, pp. 1025–1041, 1994.

[Koul 97] C. P. Koulamas. "Decomposition and Hybrid Simulated Annealing Heuristics for the Parallel–Machine Total Tardines Problem". *Naval Research Logistics*, Vol. 44, pp. 109–125, 1997.

[Koul 98] C. P. Koulamas. "A New Constructive Heuristic for the Flowshop Scheduling Problem". *European Journal of Operational Research*, Vol. 105, No. 1, pp. 66–71, 1998.

[Koza 92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[Laar 87] P. J. M. van Laarhoven and E. H. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, London, UK, 1987.

[Laum 02] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. "Combining Convergence and Diversity in Evolutionary Multi-Objective Optimization". *Evolutionary Computation*, Vol. 10, No. 3, 2002.

[Lawl 69] E. L. Lawler and C. U. Moore. "A Functional Equation and its Application to Resource Allocation and Sequencing Problems". *Management Science*, Vol. 16, pp. 77–84, 1969.

[Lawl 77] E. L. Lawler. "A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness". *Annals of Discrete Mathematics*, Vol. 1, pp. 331–342, 1977.

[Lawl 79] E. L. Lawler. "Preemptive Scheduling of Uniform Parallel Machines to Minimize the Weighted Number of Late jobs". Tech. Rep. BW 106, Centre for Mathematics and Computer Science, Amsterdam, NL, 1979.

[Lawl 93] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. "Sequencing and Scheduling Algorithms and Complexity". In: A. R. K. S.C. Graves and P. Zipkin, Eds., *Logistics of Production and Inventory*, pp. 445–522, North-Holland, Amsterdam, NL, 1993.

[Lens 90] J. K. Lenstra, D. B. Shmoys, and E. Tardos. "Approximation Algorithms for Scheduling Unrelated Parallel Machines". *Mathematical Programming*, Vol. 46, pp. 259–271, 1990.

[Lens 97] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. "Complexity of Machine Scheduling Problems". *Annals of Discrete Mathematics*, Vol. 1, pp. 343–362, 1997.

[Lour 02] H. R. Lourenço, O. Martin, and T. Stützle. "Iterated Local Search". In: F. Glover and G. Kochenberger, Eds., *Handbook of Metaheuristics*, pp. 321–353, Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Love 83] M. C. Lovell. "Data Mining". *Review of Economics and Statistics*, Vol. 65, pp. 1–12, 1983.

[Main 01] J. M. Maindonald. *Using R for Data Analysis and Graphics: An Introduction*. Statistical Consulting Unit of the Graduate School, Australian National University, Canberra, AU, 2001.

[Maro 94] O. Maron and A. W. Moore. "Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation". In: *Advances in Neural Information Processing*, pp. 59–66, Morgan Kaufman, San Francisco, CA, USA, 1994.

[Mats 87] H. Matsuo, C. J. Suh, and R. S. Sullivan. "A Controlled Search Simulated Annealing Method for the Single Machine Weighted Tardiness Problem". Working paper 87-12-2, Department of Management, University of Texas at Austin, TX, USA, 1987.

[McGe 96] C. C. McGeoch. "Toward an Experimental Method for Algorithm Simulation". *INFORMS Journal on Computing*, Vol. 8, No. 1, pp. 1–15, 1996.

[McGe 99] C. C. McGeoch and B. M. E. Moret. "How to Present a Paper on Experimental Work with Algorithms". *SIGACT News*, Vol. 30, No. 4, pp. 85–90, 1999.

[Mill 90] A. J. Miller. *Subset Selection in Regression*. Chapman & Hall, 1990.

[Mlad 97] N. Mladenović and P. Hansen. "Variable Neighborhood Search". *Computers & Operations Research*, Vol. 24, pp. 1097–1100, 1997.

[Mont 91] D. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, USA, 1991.

[Moor 68] J. M. Moore. "A n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs". *Management Science*, Vol. 15, No. 1, pp. 102–109, 1968.

[Moor 94] A. W. Moore and M. S. Lee. "Efficient Algorithms for Minimizing Cross Validation Error.". In: *International Conference on Machine Learning*, pp. 190–198, Morgan Kaufman, San Francisco, CA, USA, 1994.

[More 02] B. M. E. Moret. "Towards a Discipline of Experimental Algorithmics". In: M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, Eds., *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pp. 197–214, American Mathematical Society, 2002.

[Mort 84] T. E. Morton, R. M. Rachamadugu, and A. Vepsalainen. "Accurate Myopic Heuristics for Tardiness Scheduling". GSIA Working Paper 36, Carnegie–Mellon University, Pittsburgh, PA, USA, 1984.

[Mort 93] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems with Applications to Production Systems and Projects Management*. John Wiley & Sons, Chichester, UK, 1993.

[Nawa 83] M. Nawaz, E. Enscore, and I. Ham. "A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem". *OMEGA*, Vol. 11, No. 1, pp. 91–95, 1983.

[Nowi 96] E. Nowicki and C. Smutnicki. "A Fast Tabu Search Algorithm for the Permutation Flow-Shop Problem". *European Journal of Operational Research*, Vol. 91, pp. 160–175, 1996.

[Osma 96] I. H. Osman and J. P. Kelly. "Metaheuristics: An Overview". In: *Metaheuristics: Theory & Applications*, pp. 1–21, Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[Panw 93] S. S. Panwalkar, M. L. Smith, and C. P. Koulamas. "A Heuristic for the Single Machine Tardiness Problem". *European Journal of Operational Research*, Vol. 70, pp. 304–310, 1993.

[Park 98] M. W. Park and Y. D. Kim. "A Systematic Procedure for Setting Parameters in Simulated Annealing Algorithms". *Computers & Operations Research*, Vol. 24, No. 3, pp. 207–217, 1998.

[Pars 97] R. Parsons and J. Johnson. "A Case Study in Experimental Design Applied to Genetic ALgorithms with Applications to DNA Sequence Assembly". *American Journal of Mathematical and Management Sciences*, Vol. 17, pp. 369–396, 1997.

[Pavl 03] M. Pavlin, H. H. Hoos, and T. Stützle. "Stochastic Local Search for Multiprocessor Scheduling". In: *Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 96–113, Springer Verlag, Berlin, 2003.

[Pine 95] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs, NJ, USA, 1995.

[Pott 82] C. N. Potts and L. N. van Wassenhove. "A Decomposition Algorithm for the Single Machine Tardiness Problem". *Operations Research Letters*, Vol. 32, pp. 177–181, 1982.

[Pott 85] C. N. Potts and L. N. van Wassenhove. "A Branch and Bound Algorithm for the Total Weighted Tardiness Problem". *Operations Research*, Vol. 33, No. 2, pp. 363–377, 1985.

[Pott 91] C. N. Potts and L. N. van Wassenhove. "Single Machine Tardiness Sequencing Heuristics". *IEEE Transactions on Computers*, Vol. 23, pp. 346–354, 1991.

[Prit 69] A. A. B. Pritsker, L. J. Walters, and P. M. Wolfe. "Multiproject Scheduling With Limited Resources: A Zero-One Programming Approach". *Management Science*, Vol. 16, pp. 93–108, 1969.

[Ray 99] T. G. Ray and E. Triantaphyllou. "Procedures For the Evaluation of Conflicts in Rankings of Alternatives". *Computers and Industrial Engineering*, Vol. 36, No. 1, pp. 35–44, 1999.

[Reev 99] C. R. Reeves. "Landscapes, operators and heuristic search". *Annals of Operations Research*, Vol. 86, pp. 473–490, 1999.

[Rinn 75] A. H. G. Rinnooy Kan, B. J. Lageweg, and J. K. Lenstra. "Minimizing Total Costs in One-Machine Scheduling". *Operations Research*, Vol. 23, pp. 908–927, 1975.

[Ripl 96] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

[Robe 98] S. Robertson, B. Golden, and E. Wasil. "Neural Network Models for Initial Public Offerings". *Neurocomputing*, Vol. 18, pp. 165–182, 1998.

[Schr 03] A. Schrijver. "On the History of Combinatorial Optimization". 2003. Available from http://www.cwi.nl/∼lex.

[Sen 89] T. Sen, P. Dileepan, and J. N. D. Gupta. "The Two-Machine Flowshop Scheduling Problem With Total Tardiness". *Computers and Operations Research*, Vol. 16, No. 4, pp. 333–340, July 1989.

[Seva 01] M. Sevaux and P. Thomin. "Heuristics and Metaheursitics for a Parallel Machine Scheduling Problem: A Computational Evaluation". Tech. Rep. 2001-2, University of Valenciennes, Valenciennes, FR, November 2001.

[Seva 03] M. Sevaux and S. Dauzère-Pérès. "Genetic Algorithms to Minimize the Weighted Number of Late Jobs on a Single Machine". *European Journal of Operational Research*, Vol. 151, pp. 296–306, 2003.

[Shwi 72] J. Shwimmer. "On the N-Job One-Machine, Sequence Independent Scheduling Problem With Tardiness Penalties: A Branch-and-Bound Solution". *Management Science*, pp. 301–313, 1972.

[Sivr 99] F. Sivrikaya–Şerifoğlu and G. Ulusoy. "Parallel Machine Scheduling with Earliness and Tardiness Penalties". *Computers and Operations Research*, Vol. 26, No. 8, pp. 773–787, 1999.

[Stut 98] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, Darmstadt University of Technology, Department of Computer Science, Darmstadt, DE, 1998.

[Szwa 01] W. Szwarc, A. Grosso, and F. Della Croce. "Algorithmic Paradoxes of the Single Machine Total Tardiness Problem". *Journal of Scheduling*, Vol. 4, No. 2, pp. 93–104, 2001.

[Tail 93] E. D. Taillard. "Benchmarks for Basic Scheduling Problems". *European Journal of Operational Research*, Vol. 64, No. 2, pp. 278–285, 1993.

[Ther 97] T. M. Therneau and E. J. Atkinson. *An Introduction to Recursive Partitioning Using the RPART Routines*. 1997. Available from http://www.stats.ox.ac.uk/pub/SWin.

[Tria 98] E. Triantaphyllou, B. Shu, S. Nieto Sanchez, and T. Ray. "Multi-Criteria Decision Making: An Operations Research Approach". In: J. G. Webster, Ed., *Encyclopedia of Electrical and Electronics Engineers*, pp. 175–186, John Wiley & Sons, New York, NY, USA, 1998.

[Tria 99] E. Triantaphyllou. "Reduction of Pairwise Comparisons in Decision Making via a Duality Approach". *Multi-Criteria Decision Analysis*, Vol. 8, No. 6, pp. 299–310, July 1999. However, this issue was published on July 2000.

[Vena 99] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-Plus*. *Statistics and Computing*, Springer Verlag, New York, NY, USA, third Ed., 1999.

[Veps 85] A. Vepsalainen and T. E. Morton. "Leadtime Estimation for Priority Scheduling with Strategic Tardiness Costs". Working Paper 85-09-03, University of Pennsylvania, Philadelphia, PA, USA, 1985.

[Veps 87] A. Vepsalainen and T. E. Morton. "Priority Rules and Lead Time Estimation for Job Shop Scheduling with Weighted Tardiness Costs". *Management Science*, Vol. 33, pp. 1036–1047, 1987.

[Voli 97] C. T. Volinsky. *Bayesian Model Averaging for Censored Survival Models*. PhD thesis, University of Washington, Seattle, USA, 1997.

[Wats 02] J.-P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe. "Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems: Search Space Topology and Algorithm Performance". Group Paper, GENITOR, Colorado State University, Fort Collins, CO, USA, 2002. Available from http://www.cs.colostate.edu/∼genitor/.

[Wats 03] J.-P. Watson, J. C. Beck, A. E. Howe, and L. D. Whitley. "Problem Difficulty for Tabu Search in Job-Shop Scheduling". *Artificial Intelligence*, Vol. 143, No. 2, February 2003.

[Wilk 71] L. J. Wilkerson and J. D. Irwin. "An Improved Algorithm for Scheduling Independent Tasks". *AIIE Transactions*, Vol. 3, pp. 239–245, 1971.

[Xu 98] J. Xu, S. Chiu, and F. Glover. "Fine-tuning a Tabu Search Algorithm with Statistical Tests". *International Transactions in Operational Research*, Vol. 5, No. 3, pp. 233–244, 1998.

[Zitz 02] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. "Performance Assessment of Multiobjective Optimizers: An Analysis and Review". *IEEE Transactions on Evolutionary Computation*, Vol. 7, No. 2, pp. 117–132, 2002.

Index